Sun Studio 12 Update 1: C++
User's Guide
Contents

2.4.6 Understanding the Compiler Organization ......................................................... 39
2.5 Preprocessing Directives and Names ..................................................................... 40
  2.5.1 Pragmas ............................................................................................................ 40
  2.5.2 Macros With a Variable Number of Arguments ............................................. 41
  2.5.3 Predefined Names ............................................................................................... 41
  2.5.4 #error .............................................................................................................. 41
2.6 Memory Requirements ........................................................................................... 42
  2.6.1 Swap Space Size ............................................................................................... 42
  2.6.2 Increasing Swap Space .................................................................................... 42
  2.6.3 Control of Virtual Memory ............................................................................. 42
  2.6.4 Memory Requirements .................................................................................... 43
2.7 Using the strip Command with C++ Objects ....................................................... 44
2.8 Simplifying Commands ......................................................................................... 44
  2.8.1 Using Aliases Within the C Shell .................................................................... 44
  2.8.2 Using CCFLAGS to Specify Compile Options .............................................. 44
  2.8.3 Using make ..................................................................................................... 45

3 Using the C++ Compiler Options .......................................................................... 47
  3.1 Syntax .................................................................................................................. 47
  3.2 General Guidelines .............................................................................................. 48
  3.3 Options Summarized by Function ...................................................................... 48
    3.3.1 Code Generation Options ............................................................................ 48
    3.3.2 Compile-Time Performance Options ......................................................... 49
    3.3.3 Compile-Time and Link-Time Options ....................................................... 50
    3.3.4 Debugging Options ...................................................................................... 51
    3.3.5 Floating-Point Options ................................................................................. 52
    3.3.6 Language Options ....................................................................................... 52
    3.3.7 Library Options ............................................................................................ 53
    3.3.8 Obsolete Options ......................................................................................... 54
    3.3.9 Output Options ............................................................................................. 55
    3.3.10 Run-Time Performance Options ............................................................... 56
    3.3.11 Preprocessor Options ............................................................................... 58
    3.3.12 Profiling Options ....................................................................................... 58
    3.3.13 Reference Options .................................................................................... 59
    3.3.14 Source Options .......................................................................................... 59
6.2.2 Class Template Definition ................................................................. 77
6.2.3 Class Template Member Definitions .................................................. 77
6.2.4 Class Template Use ........................................................................... 78
6.3 Template Instantiation ......................................................................... 79
   6.3.1 Implicit Template Instantiation .......................................................... 79
   6.3.2 Explicit Template Instantiation ......................................................... 79
6.4 Template Composition ........................................................................... 80
6.5 Default Template Parameters ............................................................... 81
6.6 Template Specialization ......................................................................... 81
   6.6.1 Template Specialization Declaration ................................................ 81
   6.6.2 Template Specialization Definition ................................................... 81
   6.6.3 Template Specialization Use and Instantiation ................................... 82
   6.6.4 Partial Specialization ......................................................................... 82
6.7 Template Problem Areas ....................................................................... 83
   6.7.1 Nonlocal Name Resolution and Instantiation .................................... 83
   6.7.2 Local Types as Template Arguments ................................................. 84
   6.7.3 Friend Declarations of Template Functions ....................................... 84
   6.7.4 Using Qualified Names Within Template Definitions ....................... 86
   6.7.5 Nesting Template Names .................................................................. 87
   6.7.6 Referencing Static Variables and Static Functions ......................... 87
   6.7.7 Building Multiple Programs Using Templates in the Same Directory .... 87

7 Compiling Templates .................................................................................. 91
7.1 Verbose Compilation ............................................................................. 91
7.2 Repository Administration ...................................................................... 91
   7.2.1 Generated Instances ....................................................................... 91
   7.2.2 Whole-Class Instantiation ................................................................. 92
   7.2.3 Compile-Time Instantiation ............................................................... 92
   7.2.4 Template Instance Placement and Linkage ..................................... 92
7.3 External Instances ................................................................................ 93
   7.3.1 Possible Cache Conflicts ................................................................. 94
   7.3.2 Static Instances ............................................................................... 94
   7.3.3 Global Instances ............................................................................ 95
   7.3.4 Explicit Instances .......................................................................... 95
   7.3.5 Semi-Explicit Instances ................................................................. 96
7.4 The Template Repository ....................................................................................................... 96
  7.4.1 Repository Structure ....................................................................................................... 96
  7.4.2 Writing to the Template Repository .............................................................................. 96
  7.4.3 Reading From Multiple Template Repositories .......................................................... 97
  7.4.4 Sharing Template Repositories ....................................................................................... 97
  7.4.5 Template Instance Automatic Consistency With -instances=extern .......................... 97
7.5 Template Definition Searching ............................................................................................ 98
  7.5.1 Source File Location Conventions ................................................................................ 98
  7.5.2 Definitions Search Path ................................................................................................ 98
  7.5.3 Troubleshooting a Problematic Search .......................................................................... 98

8 Exception Handling .................................................................................................................. 101
  8.1 Synchronous and Asynchronous Exceptions ....................................................................... 101
  8.2 Specifying Runtime Errors ................................................................................................ 101
  8.3 Disabling Exceptions ......................................................................................................... 102
  8.4 Using Runtime Functions and Predefined Exceptions ...................................................... 102
  8.5 Mixing Exceptions With Signals and Setjmp/Longjmp .................................................... 104
  8.6 Building Shared Libraries That Have Exceptions ............................................................ 104

9 Cast Operations ....................................................................................................................... 105
  9.1 const_cast .......................................................................................................................... 105
  9.2 reinterpret_cast .................................................................................................................. 106
  9.3 static_cast .......................................................................................................................... 107
  9.4 Dynamic Casts ..................................................................................................................... 107
    9.4.1 Casting Up the Hierarchy .............................................................................................. 108
    9.4.2 Casting to void* .......................................................................................................... 108
    9.4.3 Casting Down or Across the Hierarchy ....................................................................... 108

10 Improving Program Performance ............................................................................................ 111
  10.1 Avoiding Temporary Objects ........................................................................................... 111
  10.2 Using Inline Functions ..................................................................................................... 112
  10.3 Using Default Operators .................................................................................................. 112
  10.4 Using Value Classes ......................................................................................................... 113
    10.4.1 Choosing to Pass Classes Directly .............................................................................. 114
12.6 Using Shared Libraries ................................................................. 148
12.7 Replacing the C++ Standard Library .............................................. 148
  12.7.1 What Can Be Replaced ............................................................. 149
  12.7.2 What Cannot Be Replaced ........................................................ 149
  12.7.3 Installing the Replacement Library ......................................... 149
  12.7.4 Using the Replacement Library .............................................. 149
  12.7.5 Standard Header Implementation ......................................... 150

13 Using The C++ Standard Library ...................................................... 153
  13.1 C++ Standard Library Header Files ........................................... 154
  13.2 C++ Standard Library Man Pages ............................................ 155
  13.3 STLport .................................................................................. 168
    13.3.1 Redistribution and Supported STLport Libraries .................. 169

14 Using the Classic iostream Library .................................................. 171
  14.1 Predefined iostreams ............................................................ 171
  14.2 Basic Structure of iostream Interaction ...................................... 172
  14.3 Using the Classic iostream Library ........................................... 173
    14.3.1 Output Using iostream .................................................. 173
    14.3.2 Input Using iostream .................................................. 176
    14.3.3 Defining Your Own Extraction Operators ......................... 176
    14.3.4 Using the char* Extractor ........................................... 177
    14.3.5 Reading Any Single Character ....................................... 177
    14.3.6 Binary Input ............................................................. 178
    14.3.7 Peeking at Input ........................................................ 178
    14.3.8 Extracting Whitespace ................................................ 178
    14.3.9 Handling Input Errors ................................................ 179
    14.3.10 Using iostreams With stdio ....................................... 179
  14.4 Creating iostreams ............................................................... 180
    14.4.1 Dealing With Files Using Class fstream .......................... 180
    14.5 Assignment of iostreams .................................................... 183
    14.6 Format Control ............................................................... 183
    14.7 Manipulators .................................................................. 183
      14.7.1 Using Plain Manipulators ....................................... 185
      14.7.2 Parameterized Manipulators .................................. 186
Part IV  Appendixes ........................................................................................................................................209

A  C++ Compiler Options ......................................................................................................................................211
   A.1 How Option Information Is Organized ..............................................................................................................211
   A.2 Option Reference ..................................................................................................................................................212
      A.2.1 –386 .................................................................................................................................................................212
      A.2.2 –486 .................................................................................................................................................................212
      A.2.3 –a ......................................................................................................................................................................213
      A.2.4 –binding .........................................................................................................................................................213
      A.2.5 –c ......................................................................................................................................................................214
      A.2.6 –cg[89|92] ........................................................................................................................................................215
      A.2.7 –compat=[4|5] ................................................................................................................................................215
      A.2.8 +d ......................................................................................................................................................................216
      A.2.9 -dname=[def] ..................................................................................................................................................217
      A.2.10 –dy[n] ..........................................................................................................................................................219
      A.2.11 –dalign .........................................................................................................................................................220
      A.2.12 –dryrun .......................................................................................................................................................220
      A.2.13 –E ..................................................................................................................................................................221
      A.2.14 +e[0|1] ............................................................................................................................................................222
      A.2.15 -erroff=[r] ....................................................................................................................................................222
      A.2.16 -errtags=[a] ..................................................................................................................................................223
      A.2.17 -errwarn=[r] ...............................................................................................................................................224
      A.2.18 –fast ...............................................................................................................................................................225
      A.2.19 –features=a[,a...] ...................................................................................................................................227
      A.2.20 -filt=[filter[,filter...]] ..................................................................................................................................231
      A.2.21 –flags ............................................................................................................................................................233
      A.2.22 -fma=[none|fused] ......................................................................................................................................234
      A.2.23 –fnonstd ........................................................................................................................................................234
      A.2.24 -fns=[yes|no] ...............................................................................................................................................234
      A.2.25 -fprecision=p .............................................................................................................................................236
      A.2.26 -fround=r ....................................................................................................................................................237
      A.2.27 -fsimple=[n] ..............................................................................................................................................238
      A.2.28 -fstore ..........................................................................................................................................................239
      A.2.29 -ftrap=[r,...] ...............................................................................................................................................240
      A.2.30 –G .................................................................................................................................................................241
| A.2.31 | -g | ......................................................................................................................... | 242 |
| A.2.32 | -g0 | ....................................................................................................................... | 243 |
| A.2.33 | -H | ....................................................................................................................... | 244 |
| A.2.34 | -h| [ ]name | ....................................................................................................................... | 244 |
| A.2.35 | -help | ................................................................................................................. | 244 |
| A.2.36 | -Ipathname | ....................................................................................................................... | 244 |
| A.2.37 | -I | ........................................................................................................................... | 251 |
| A.2.38 | -i | ....................................................................................................................... | 251 |
| A.2.39 | -include filename | ....................................................................................................................... | 248 |
| A.2.40 | -inline | ....................................................................................................................... | 248 |
| A.2.41 | -instances=a | ....................................................................................................................... | 248 |
| A.2.42 | -instlib=filename | ....................................................................................................................... | 249 |
| A.2.43 | -KPIC | ....................................................................................................................... | 250 |
| A.2.44 | -Kpic | ....................................................................................................................... | 251 |
| A.2.45 | -keeptmp | ....................................................................................................................... | 251 |
| A.2.46 | -Lpath | ....................................................................................................................... | 251 |
| A.2.47 | -lib | ....................................................................................................................... | 251 |
| A.2.48 | -libmieee | ....................................................................................................................... | 252 |
| A.2.49 | -libmil | ....................................................................................................................... | 252 |
| A.2.50 | -library=| [ ][,][... ] | ................................................................................................................. | 252 |
| A.2.51 | -m32| -m64 | ....................................................................................................................... | 256 |
| A.2.52 | -mc | ....................................................................................................................... | 257 |
| A.2.53 | -migration | ....................................................................................................................... | 257 |
| A.2.54 | -misalign | ....................................................................................................................... | 257 |
| A.2.55 | -mr[,[string]] | ....................................................................................................................... | 258 |
| A.2.56 | -mt | ....................................................................................................................... | 258 |
| A.2.57 | -native | ....................................................................................................................... | 259 |
| A.2.58 | -noex | ....................................................................................................................... | 259 |
| A.2.59 | -nofstore | ....................................................................................................................... | 259 |
| A.2.60 | -nolib | ....................................................................................................................... | 259 |
| A.2.61 | -nolibmil | ....................................................................................................................... | 259 |
| A.2.62 | -noqueue | ....................................................................................................................... | 259 |
| A.2.63 | -norunpath | ....................................................................................................................... | 259 |
| A.2.64 | -O | ....................................................................................................................... | 260 |
| A.2.65 | -Olevel | ....................................................................................................................... | 260 |
| A.2.66 | -o| filename | ....................................................................................................................... | 260 |
A.2.67 +p ................................................................. 261
A.2.68 –P .................................................................. 261
A.2.69 –p .................................................................. 261
A.2.70 –pentium ..................................................... 261
A.2.71 –pg ................................................................ 262
A.2.72 –PIC ............................................................. 262
A.2.73 –pic ................................................................ 262
A.2.74 –pta ............................................................... 262
A.2.75 –ptipath ......................................................... 262
A.2.76 –pto ................................................................ 262
A.2.77 –ptr ................................................................ 263
A.2.78 –ptv ................................................................ 263
A.2.79 –Qoption phase option[+option...] .................. 263
A.2.80 –qoption phase option .................................... 264
A.2.81 –qp ................................................................ 264
A.2.82 –qproduce sourcetype ................................. 265
A.2.83 –qproduce sourcetype ...................................... 265
A.2.84 –Rpathname[:pathname...] ......................... 265
A.2.85 –readme ........................................................ 266
A.2.86 –S .................................................................. 266
A.2.87 –s .................................................................. 266
A.2.88 –sb ................................................................ 266
A.2.89 –sbfast .......................................................... 266
A.2.90 –staticlib=[l,[...]] ......................................... 266
A.2.91 –sync_stdio=[yes|no] ...................................... 268
A.2.92 –temp=path .................................................... 269
A.2.93 –template=opt[,[opt...]] ............................... 269
A.2.94 –time .............................................................. 271
A.2.95 –Uname .......................................................... 271
A.2.96 –unroll=n ...................................................... 271
A.2.97 –V ................................................................ 272
A.2.98 –v .................................................................. 272
A.2.99 –vdelx ............................................................. 272
A.2.100 –verbose=v[,...] ......................................... 272
A.2.101 +w ............................................................... 273
A.2.102 +w2 ............................................................. 274
<table>
<thead>
<tr>
<th>Option</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>–w</code></td>
<td>274</td>
</tr>
<tr>
<td><code>–xm</code></td>
<td>274</td>
</tr>
<tr>
<td><code>–xa</code></td>
<td>274</td>
</tr>
<tr>
<td><code>–addr32</code></td>
<td>275</td>
</tr>
<tr>
<td><code>–xalias_level=n</code></td>
<td>275</td>
</tr>
<tr>
<td>`–xannotate=yes</td>
<td>no]`</td>
</tr>
<tr>
<td><code>–xar</code></td>
<td>278</td>
</tr>
<tr>
<td>`–xannotate=prepare</td>
<td>off`</td>
</tr>
<tr>
<td>`–xbuiltin=%all</td>
<td>none]`</td>
</tr>
<tr>
<td><code>–xcache=c</code></td>
<td>287</td>
</tr>
<tr>
<td>`–xcg=[89</td>
<td>92]`</td>
</tr>
<tr>
<td><code>–xchar=[a]</code></td>
<td>289</td>
</tr>
<tr>
<td><code>–xcheck=[i]</code></td>
<td>290</td>
</tr>
<tr>
<td><code>–xchip=c</code></td>
<td>290</td>
</tr>
<tr>
<td><code>–xcode=a</code></td>
<td>292</td>
</tr>
<tr>
<td><code>–xcrossfile=[n]</code></td>
<td>294</td>
</tr>
<tr>
<td>`–debugformat=[stabs</td>
<td>dwarf]`</td>
</tr>
<tr>
<td>`–depend=[yes</td>
<td>no]`</td>
</tr>
<tr>
<td><code>–dumpmacros=[value[,value]]</code></td>
<td>296</td>
</tr>
<tr>
<td><code>–xe</code></td>
<td>299</td>
</tr>
<tr>
<td>`–xF=[v</td>
<td>v...]`</td>
</tr>
<tr>
<td><code>–xhelp=flags</code></td>
<td>301</td>
</tr>
<tr>
<td><code>–xhelp=readme</code></td>
<td>301</td>
</tr>
<tr>
<td><code>–xhwcprof</code></td>
<td>301</td>
</tr>
<tr>
<td><code>–xia</code></td>
<td>302</td>
</tr>
<tr>
<td><code>–xinline=[func_spec[,func_spec...]]</code></td>
<td>302</td>
</tr>
<tr>
<td>`–xinstrument=[no]</td>
<td>datarace`</td>
</tr>
<tr>
<td>`–xipo=[0</td>
<td>1</td>
</tr>
<tr>
<td><code>–xipo_archive=[a]</code></td>
<td>307</td>
</tr>
<tr>
<td><code>–xjobs=n</code></td>
<td>308</td>
</tr>
<tr>
<td><code>–xlang=language[,language]</code></td>
<td>309</td>
</tr>
<tr>
<td><code>–xdscope=[v]</code></td>
<td>310</td>
</tr>
<tr>
<td><code>–xlibmieee</code></td>
<td>311</td>
</tr>
<tr>
<td><code>–xlibmil</code></td>
<td>311</td>
</tr>
</tbody>
</table>
A.2.139 -xlibmopt .............................................................. 312
A.2.140 -xic_lib=sunperf ................................................. 312
A.2.141 -xicinfo .............................................................. 312
A.2.142 -xlinkopt=[level] .................................................. 313
A.2.143 -xloopinfo ........................................................... 314
A.2.144 -xM ................................................................. 314
A.2.145 -xM1 ................................................................. 315
A.2.146 -xMD ................................................................. 315
A.2.147 -xMF ................................................................. 315
A.2.148 -xMMD .............................................................. 316
A.2.149 -xMerge .............................................................. 316
A.2.150 -xmaxopt=[v] ......................................................... 316
A.2.151 -xmemalign=ab .................................................... 316
A.2.152 -xmodel=[a] ........................................................ 318
A.2.153 -xnolib ............................................................. 319
A.2.154 -xnolibmil .......................................................... 320
A.2.155 -xnolibmopt ....................................................... 320
A.2.156 -xnorunpath ......................................................... 321
A.2.157 -xOlevel ............................................................ 321
A.2.158 -xopenmp=[i] ....................................................... 324
A.2.159 -xpagesize=n ..................................................... 325
A.2.160 -xpagesize_heap=n ............................................. 326
A.2.161 -xpagesize_stack=n ............................................ 327
A.2.162 -xpch=v ............................................................ 328
A.2.163 -xpchstop=file .................................................... 331
A.2.164 -xspec=[{yes|no}] ............................................... 331
A.2.165 -xpg ............................................................... 331
A.2.166 -xport64=[(v)] .................................................... 332
A.2.167 -xprefetch=[a,[a...]] ........................................... 335
A.2.168 -xprefetch_auto_type=a ...................................... 338
A.2.169 -xprefetch_level=[i] ............................................ 338
A.2.170 -xprofile=p ....................................................... 339
A.2.171 -xprofile_ircache=[path] ..................................... 342
A.2.172 -xprofile_pathmap ............................................... 342
A.2.173 -xreduction ....................................................... 343
A.2.174 -xregs=[r,[r...]] ................................................ 343

Contents
B Pragmas ......................................................................................................................... 363
B.1 Pragma Forms ............................................................................................................. 363
B.1.1 Overloaded Functions as Pragma Arguments .......................................................... 363
B.2 Pragma Reference ......................................................................................................... 364
B.2.1 #pragma align ........................................................................................................... 364
B.2.2 #pragma does_not_read_global_data .................................................................... 365
B.2.3 #pragma does_not_return ....................................................................................... 365
B.2.4 #pragma does_not_write_global_data ................................................................. 366
B.2.5 #pragma dumpmacros ............................................................................................. 366
B.2.6 #pragma end_dumpmacros ....................................................................................... 367
B.2.7 #pragma fini ............................................................................................................... 368
B.2.8 #pragma hdrstop .................................................................................................... 368
B.2.9 #pragma ident ............................................................................................................ 368
B.2.10 #pragma init ............................................................................................................ 369
B.2.11 #pragma must_have_frame ................................................................................... 369
B.2.12 #pragma no_side_effect ...................................................................................... 369
Examples

EXAMPLE 6–1  Example of Local Type as Template Argument Problem ......................... 84
EXAMPLE 6–2  Example of Friend Declaration Problem ................................................................. 85
EXAMPLE 11–1  Checking Error State .................................................................................. 124
EXAMPLE 11–2  Calling gcount ............................................................................................ 125
EXAMPLE 11–3  User-Defined I/O Operations ....................................................................... 125
EXAMPLE 11–4  Disabling MT-Safety .................................................................................... 126
EXAMPLE 11–5  Switching to MT-Unsafe ............................................................................. 126
EXAMPLE 11–6  Using Synchronization With MT-Unsafe Objects ..................................... 127
EXAMPLE 11–7  New Classes ................................................................................................. 127
EXAMPLE 11–8  New Class Hierarchy ..................................................................................... 128
EXAMPLE 11–9  New Functions ............................................................................................... 128
EXAMPLE 11–10 Example of Using Locking Operations ...................................................... 131
EXAMPLE 11–11 Making I/O Operation and Error Checking Atomic ..................................... 132
EXAMPLE 11–12 Destroying a Shared Object ........................................................................... 133
EXAMPLE 11–13 Using ofstream Objects in an MT-Safe Way ............................................. 134
EXAMPLE 14–1 string Extraction Operator ............................................................................. 177
EXAMPLE A–1 Preprocessor Example Program foo.cc ........................................................... 221
EXAMPLE A–2 Preprocessor Output of foo.cc Using -E Option .............................................. 221
EXAMPLE A–3 A Loop With Two Pointers ............................................................................. 347
Preface

The Sun Studio 12 Update 1 C++ User's Guide guide describes the environment and command-line options for the Sun Studio C++ compiler, CC.

This guide is intended for application developers with a working knowledge of the C++ language and Solaris or Linux operating environments who want to learn how to use the Sun Studio C++ compiler effectively.

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AaBbCc123</td>
<td>The names of commands, files, and directories, and onscreen computer output</td>
<td>Edit your .login file. Use ls -a to list all files. machine_name% you have mail.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>What you type, contrasted with onscreen computer output</td>
<td>machine_name% su Password:</td>
</tr>
<tr>
<td>aabbcc123</td>
<td>Placeholder: replace with a real name or value</td>
<td>The command to remove a file is rm filename.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Book titles, new terms, and terms to be emphasized</td>
<td>Read Chapter 6 in the User's Guide. A cache is a copy that is stored locally. Do not save the file. Note: Some emphasized items appear bold online.</td>
</tr>
</tbody>
</table>
Shell Prompts in Command Examples

The following table shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

**TABLE P-2  Shell Prompts**

<table>
<thead>
<tr>
<th>Shell</th>
<th>Prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td>C shell</td>
<td>machine_name%</td>
</tr>
<tr>
<td>C shell for superuser</td>
<td>machine_name#</td>
</tr>
<tr>
<td>Bourne shell and Korn shell</td>
<td>$</td>
</tr>
<tr>
<td>Bourne shell and Korn shell for superuser</td>
<td>#</td>
</tr>
</tbody>
</table>

Supported Platforms

This Sun™ Studio release supports systems that use the SPARC and x86 families of processor architectures: UltraSPARC®, SPARC64, AMD64, Pentium, and Xeon EM64T. The supported systems for the version of the Solaris Operating System you are running are available in the hardware compatibility lists at [http://www.sun.com/bigadmin/hcl](http://www.sun.com/bigadmin/hcl). These documents cite any implementation differences between the platform types.

In this document, these x86 related terms mean the following:

- “x86” refers to the larger family of 64–bit and 32–bit x86 compatible products.
- “x64” points out specific 64–bit information about AMD64 or EM64T systems.
- “32–bit x86” points out specific 32–bit information about x86 based systems.

For supported systems, see the hardware compatibility lists.

Accessing Sun Studio Documentation

You can access the documentation at the following locations:

- Online help for all components of the IDE is available through the Help menu, as well as through Help buttons on many windows and dialog boxes, in the IDE.
- Online help for the Performance Analyzer is available through the Help menu, as well as through Help buttons on many windows and dialog boxes, in the Performance Analyzer.

The [docs.sun.com](http://docs.sun.com) web site ([http://docs.sun.com](http://docs.sun.com)) enables you to read, print, and buy Sun Microsystems manuals through the Internet.
**Note** – Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

### Documentation in Accessible Formats

The documentation is provided in accessible formats that are readable by assistive technologies for users with disabilities. You can find accessible versions of documentation as described in the following table.

<table>
<thead>
<tr>
<th>Type of Documentation</th>
<th>Format and Location of Accessible Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manuals</td>
<td>HTML at <a href="http://docs.sun.com">http://docs.sun.com</a></td>
</tr>
<tr>
<td>Readmes</td>
<td>HTML on the developer portal at <a href="http://developers.sun.com/sunstudio/documentation/ss12u1/">http://developers.sun.com/sunstudio/documentation/ss12u1/</a></td>
</tr>
<tr>
<td>Man pages</td>
<td>HTML on the developer portal at <a href="http://developers.sun.com/sunstudio/documentation/ss12u1/">http://developers.sun.com/sunstudio/documentation/ss12u1/</a></td>
</tr>
<tr>
<td>Online help</td>
<td>HTML available through the Help menu and Help buttons in the IDE</td>
</tr>
<tr>
<td>Release notes</td>
<td>HTML at <a href="http://docs.sun.com">http://docs.sun.com</a></td>
</tr>
</tbody>
</table>

### Resources for Developers

Visit [http://developers.sun.com/sunstudio](http://developers.sun.com/sunstudio) to find these frequently updated resources:

- Articles on programming techniques and best practices
- Documentation of the software, as well as corrections to the documentation that is installed with your software
- Tutorials that take you step-by-step through development tasks using Sun Studio tools
- Information on support levels
- User forums
- Downloadable code samples
- New technology previews
The Sun Studio portal is one of a number of additional resources for developers at the Sun Developer Network web site, http://developers.sun.com.

Contacting Technical Support

If you have technical questions about this product that are not answered in this document, go to http://www.sun.com/service/contacting

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Submit your comments to Sun at this URL: http://www.sun.com/hwdocs/feedback.

Please include the part number of the document in the subject line of your email. For example, the part number for this document is 820-7599-10.
1.1 New Features and Functionality of the Sun Studio 12 Update 1 C++ 5.10 Compiler

This section provides a brief overview of the new C compiler features and functionality introduced in the Sun Studio 12 Update 1 C++ 5.10 Compiler release. For specific explanations, follow the cross references provided for each item.

- Object files created by the compilers on the Solaris OS on x86 platforms or the Linux OS are incompatible with previous compiler versions if the application code contains functions with parameters or return values using _m128/_m64 data types. Users with .il inline function files, assembler code, or asm inline statements calling these functions also need to be aware of this incompatibility.
- New -xtarget values for x86 processors woodcrest, penryn, nehalem, core2; and SPARC processors ultraT2plus and sparc64vii.
- New -xarch values for x86 architectures ssse3, sse4_1, sse4_2, and for SPARC architectures sparcima.
- New -xchip values for SPARC processors sparc64vii, ultraT2plus. New -xchip values for x86 processors core2, penryn, nehalem.
- The -xprofile=collect and -xprofile=use options provide improved support for profiling multi-threaded, dynamically linked applications.
- The -xcrossfile=1 option becomes an alias of the -xi=1 option. The -xcrossfile=0 option no longer has any effect. Specifically, -xcrossfile=1 and -xcrossfile=0 are equivalent to -xi=1.
- On Solaris platforms, the -xspec[=yes|no] option generates a PEC binary that is recompilable for use with the Automatic Tuning System (ATS).
- The -Y option does not accept i as an argument.
- On SPARC platforms, the -xdepend option is now implicitly enabled for optimization levels -x03 or higher, and is no longer included in the expansion of the -fast option.
- Support for OpenMP 3.0 includes a libmtsk library. OpenMP programs will link with this library by default instead of the libmtsk library in the Solaris OS.
- -xannotate={yes|no} (SPARC platforms only) instructs the compiler to create binaries that can be transformed later by binary modification tools like binopt(1).
- The -xia (interval arithmetic) option is now supported on x86 Solaris platforms.
- The -xipo_archive option is now supported on x86 Solaris and Linux platforms.
- The -00 option option no longer accepts ube_ipa as an argument.
- The expansion of the -fast option now includes -D MATHERR_ERRNO_DONTCARE.
- The -xvpara option, which shows parallelization warning messages, is now supported.
- The -sb, -sbfast, -xsb and -xsbfast options are now deprecated and are ignored silently.
- The compiler now inlines code when you specify the -g option with any -O or -xO optimization levels as long as you do not also specify +d.
- The pragma must_have_frame is now supported.
- In standard C++, a case label in a switch statement can have only one associated value. The Sun Studio C++ compiler allows an extension found in some compilers known as case ranges.
- The compiler normally creates temporary files in the /tmp directory. You can specify another directory by setting the TMPDIR environment variable.
- The following attributes of functions are now supported: _attribute__((const)) _attribute__((constructor)) _attribute__((destructor))
- The following attribute of variables is now supported for struct and enum types only: _attribute__((packed))
- Universal Character Names are now supported.
- Loop pragmas are now supported.
- User-defined names for macro variadic arguments are now supported.
- The -include filename option for specifying preprocessor include files has been added.
1.2 Special x86 Notes

There are some important issues to be aware of when compiling for x86 Solaris platforms.

The legacy Sun-style parallelization pragmas are not available on x86. Use OpenMP instead. See the Sun Studio 12: Update 1 OpenMP API User’s Guide for information on converting legacy parallelization directives to OpenMP.

Programs compiled with -xarch set to sse, sse2, sse2a, sse3, or beyond must be run only on platforms that provide these extensions and features.

Solaris OS releases starting with Solaris 9 4/04 are SSE/SSE2-enabled on Pentium 4-compatible platforms. Earlier versions of Solaris OS are not SSE/SSE2-enabled. If an instruction set selected by -xarch is not enabled in the running Solaris OS, the compiler will not be able to generate or link code for that instruction set.

If you compile and link in separate steps, always link using the compiler and with same -xarch setting to ensure that the correct startup routine is linked.

Numerical results on x86 might differ from results on SPARC due to the x86 80-bit floating-point registers. To minimize these differences, use the -fstore option or compile with -xarch=sse2 if the hardware supports SSE2.

Numerical results can also differ between Solaris and Linux because the intrinsic math libraries (for example, \( \sin(x) \)) are not the same.

1.3 Compiling for 64–Bit Platforms

Use the -m32 option to compile for the ILP32 32–bit model. Use the -m64 option to compile for the LP64 64–bit model.

The ILP32 model specifies that C++-language int, long, and pointer data types are all 32-bits wide. The LP64 model specifies that long and pointer data types are all 64-bits wide. The Solaris and Linux OS also support large files and large arrays under the LP64 memory model.

When you compile with -m64, the resulting executable works only on 64-bit UltraSPARC or x86 processors under Solaris OS or Linux OS running a 64-bit kernel. Compilation, linking, and execution of 64-bit objects can only take place in a Solaris or Linux OS that supports 64-bit execution.
1.4 Binary Compatibility Verification

On Solaris systems, beginning with Sun Studio 11, program binaries compiled with the Sun Studio compilers are marked with architecture hardware flags indicating the instruction sets assumed by the compiled binary. At runtime these marker flags are checked to verify that the binary can run on the hardware it is attempting to execute on.

If a program does not contain these architecture hardware flags, or if the platform does not enable the appropriate features or instruction set extensions, running the program could result in segmentation faults or incorrect results without any explicit warning messages.

This warning extends also to programs that employ .i1 inline assembly language functions or __asm() assembler code that utilize SSE, SSE2, SSE2a, and SSE3 and newer instructions and extensions.

1.5 Standards Conformance


In this document, “Standard” means conforming to the versions of the standards listed above. “Nonstandard” or “Extension” refers to features that go beyond these versions of these standards.

The responsible standards bodies may revise these standards from time to time. The versions of the applicable standards to which the C++ compiler conforms may be revised or replaced, resulting in features in future releases of the Sun C++ compiler that create incompatibilities with earlier releases.

1.6 C++ Readme File

The C++ compiler’s readme file highlights important information about the compiler, including:

- Information discovered after the manuals were printed
- New and changed features
- Software corrections
Problems and workarounds
Limitations and incompatibilities
Shippable libraries
Standards not implemented

The C++ compiler readme for this and previous Sun Studio releases can be found on the Sun Developer Network Sun Studio portal, http://developers.sun.com/sunstudio/documentation.

1.7 Man Pages

Online manual (man) pages provide immediate documentation about a command, function, subroutine, or collection of such things.

You can display a man page by running the command:

example% man topic

Throughout the C++ documentation, man page references appear with the topic name and man section number: CC(1) is accessed with man CC. Other sections, denoted by ieee_flags(3M) for example, are accessed using the -s option on the man command:

example% man -s 3M ieee_flags

1.8 Native-Language Support

This release of C++ supports the development of applications in languages other than English, including most European languages and Japanese. As a result, you can easily switch your application from one native language to another. This feature is known as internationalization.

In general, the C++ compiler implements internationalization as follows:

- C++ recognizes ASCII characters from international keyboards (in other words, it has keyboard independence and is 8-bit clean).
- C++ allows the printing of some messages in the native language.
- C++ allows native-language characters in comments, strings, and data.
- C++ supports only Extended UNIX Character (EUC) compliant character sets - a character set in which every null byte in a string is the null character and every byte in the string with the ascii value of '/ ‘ is the ’/ ’ character.

Variable names cannot be internationalized and must be in the English character set.

You can change your application from one native language to another by setting the locale. For information on this and other native-language support features, see the operating system documentation.
This chapter describes how to use the C++ compiler.

The principal use of any compiler is to transform a program written in a high-level language like C++ into a data file that is executable by the target computer hardware. You can use the C++ compiler to:

- Transform source files into relocatable binary (.o) files, to be linked later into an executable file, a static (archive) library (.a) file (using `-xar`), or a dynamic (shared) library (.so) file
- Link or relink object files or library files (or both) into an executable file
- Compile an executable file with runtime debugging enabled (`-g`)
- Compile an executable file with runtime statement or procedure-level profiling (`-pg`)

### 2.1 Getting Started

This section gives you a brief overview of how to use the C++ compiler to compile and run C++ programs. See “16.8 Using `dlopen` to Access a C++ Library From a C Program” on page 207 for a full reference to the command-line options.

**Note** – The command-line examples in this chapter show `CC` usages. Printed output might be slightly different.

The basic steps for building and running a C++ program involve:

1. Using an editor to create a C++ source file with one of the valid suffixes listed in Table 2–1
2. Invoking the compiler to produce an executable file
3. Launching the program into execution by typing the name of the executable file

The following program displays a message on the screen:
In this example, CC compiles the source file greetings.cc and, by default, compiles the executable program onto the file, a.out. To launch the program, type the name of the executable file, a.out, at the command prompt.

Traditionally, UNIX compilers name the executable file a.out. It can be awkward to have each compilation write to the same file. Moreover, if such a file already exists, it will be overwritten the next time you run the compiler. Instead, use the -o compiler option to specify the name of the executable output file, as in the following example:

```
example% CC – o greetings greetings.cc
```

In this example, the -o option tells the compiler to write the executable code to the file greetings. (It is common to give a program consisting of a single source file the name of the source file without the suffix.)

Alternatively, you could rename the default a.out file using the mv command after each compilation. Either way, run the program by typing the name of the executable file:

```
example% ./greetings
Real programmers write C++!
example%
```

### 2.2 Invoking the Compiler

The remainder of this chapter discuss the conventions used by the CC command, compiler source line directives, and other issues concerning the use of the compiler.

#### 2.2.1 Command Syntax

The general syntax of a compiler command line is as follows:

```
CC [options] [source-files] [object-files] [libraries]
```

An *option* is an option keyword prefixed by either a dash (-) or a plus sign (+). Some options take arguments.
In general, the processing of the compiler options is from left to right, allowing selective overriding of macro options (options that include other options). In most cases, if you specify the same option more than once, the rightmost assignment overrides and there is no accumulation. Note the following exceptions:

- All linker options and the -features, -f, -L, -library, -pti, -R, -staticlib, -U, -verbose, -xdumpmacros, and -xprefetch options accumulate, they do not override.

- All -U options are processed after all -D options.

Source files, object files, and libraries are compiled and linked in the order in which they appear on the command line.

In the following example, CC is used to compile two source files (growth.C and fft.C) to produce an executable file named growth with runtime debugging enabled:

```
example% CC -g -o growth growth.C fft.C
```

### 2.2.2 File Name Conventions

The suffix attached to a file name appearing on the command line determines how the compiler processes the file. A file name with a suffix other than those listed in the following table, or without a suffix, is passed to the linker.

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Language</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>.c</td>
<td>C++</td>
<td>Compile as C++ source files, put object files in current directory; default name of object file is that of the source but with an .o suffix.</td>
</tr>
<tr>
<td>.C</td>
<td>C++</td>
<td>Same action as .c suffix.</td>
</tr>
<tr>
<td>.cc</td>
<td>C++</td>
<td>Same action as .c suffix.</td>
</tr>
<tr>
<td>.cpp</td>
<td>C++</td>
<td>Same action as .c suffix.</td>
</tr>
<tr>
<td>.cxx</td>
<td>C++</td>
<td>Same action as .c suffix.</td>
</tr>
<tr>
<td>.c++</td>
<td>C++</td>
<td>Same action as .c suffix.</td>
</tr>
<tr>
<td>.i</td>
<td>C++</td>
<td>Preprocessor output file treated as C++ source file. Same action as .c suffix.</td>
</tr>
<tr>
<td>.s</td>
<td>Assembler</td>
<td>Assemble source files using the assembler.</td>
</tr>
<tr>
<td>.S</td>
<td>Assembler</td>
<td>Assemble source files using both the C language preprocessor and the assembler.</td>
</tr>
</tbody>
</table>
2.3 Compiling With Different Compiler Versions

2.2.3 Using Multiple Source Files

The C++ compiler accepts multiple source files on the command line. A single source file compiled by the compiler, together with any files that it directly or indirectly supports, is referred to as a compilation unit. C++ treats each source as a separate compilation unit.

2.3 Compiling With Different Compiler Versions

This compiler does not use the cache by default. It only uses the cache if you specify -instances=extern. If the compiler makes use of the cache, it checks the cache directory’s version and issues error messages whenever it encounters cache version problems. Future C++ compilers will also check cache versions. For example, a future compiler that has a different template cache version identification and that processes a cache directory produced by this release of the compiler might issue an error that is similar to the following message:

Template Database at ./SunWS_cache is incompatible with this compiler

Similarly, the compiler issues an error if it encounters a cache directory that was produced by a later version of the compiler.

When you upgrade your compiler, it is always good practice to clean the cache. Run CCadmin -clean on every directory that contains a template cache directory (in most cases, a template cache directory is named SunWS_cache). Alternatively, you can use rm -rf SunWS_cache.

---

**TABLE 2-1** File Name Suffixes Recognized by the C++ Compiler (Continued)

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Language</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inline expansion</td>
<td>Process assembly inline-template files for inline expansion. The compiler will use templates to expand inline calls to selected routines. (Inline-template files are special assembler files. See the inline(1) man page.)</td>
</tr>
<tr>
<td>.o</td>
<td>Object files</td>
<td>Pass object files through to the linker.</td>
</tr>
<tr>
<td>.a</td>
<td>Static (archive) library</td>
<td>Pass object library names to the linker.</td>
</tr>
<tr>
<td>.so</td>
<td>Dynamic (shared) library</td>
<td>Pass names of shared objects to the linker.</td>
</tr>
<tr>
<td>.so.ni</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.4 Compiling and Linking

This section describes some aspects of compiling and linking programs. In the following example, CC is used to compile three source files and to link the object files to produce an executable file named prgrm.

```
example% CC file1.cc file2.cc file3.cc -o prgrm
```

2.4.1 Compile-Link Sequence

In the previous example, the compiler automatically generates the loader object files (file1.o, file2.o and file3.o) and then invokes the system linker to create the executable program for the file prgrm.

After compilation, the object files (file1.o, file2.o, and file3.o) remain. This convention permits you to easily relink and recompile your files.

Note – If only one source file is compiled and a program is linked in the same operation, the corresponding .o file is deleted automatically. To preserve all .o files, do not compile and link in the same operation unless more than one source file gets compiled.

If the compilation fails, you will receive a message for each error. No .o files are generated for those source files with errors, and no executable program is written.

2.4.2 Separate Compiling and Linking

You can compile and link in separate steps. The -c option compiles source files and generates .o object files, but does not create an executable. Without the -c option, the compiler invokes the linker. By splitting the compile and link steps, a complete recompilation is not needed just to fix one file. The following example shows how to compile one file and link with others in separate steps:

```
example% CC -c file1.cc Make new object file
example% CC -o prgrm file1.o file2.o file3.o Make executable file
```

Be sure that the link step lists all the object files needed to make the complete program. If any object files are missing from this step, the link will fail with “undefined external reference” errors (missing routines).
2.4.3 Consistent Compiling and Linking

If you compile and link in separate steps, consistent compiling and linking is critical when using the compiler options listed in “3.3.3 Compile-Time and Link-Time Options” on page 50.

If you compile a subprogram using any of these options, you must link using the same option as well:

- In the case of the -library, -fast, -xtarget, and -xarch options, you must be sure to include the linker options that would have been passed if you had compiled and linked together. Use --dryrun to see the expansion of these options to determine the options needed on the link step.
- With -p, -xpg, and -xprofile, including the option in one phase and excluding it from the other phase will not affect the correctness of the program, but you will not be able to do profiling.
- With -g and -g0, including the option in one phase and excluding it from the other phase will not affect the correctness of the program, but it will affect the ability to debug the program. Any module that is not compiled with either of these options, but is linked with -g or -g0 will not be prepared properly for debugging. Note that compiling the module that contains the function main with the -g option or the -g0 option is usually necessary for debugging.

In the following example, the programs are compiled using the -library=stlport4 compiler option.

```
example% CC -library=stlport4 sbr.cc -c
example% CC -library=stlport4 main.cc -c
example% CC -library=stlport4 sbr.o main.o -o myprogram
```

If you do not use -library=stlport4 consistently, some parts of the program will use the default libCstd, and others will use the optional replacement STlport library. The resulting program might not link, and would not in any case run correctly.

If the program uses templates, it is possible that some templates will get instantiated at link time. In that case the command line options from the last line (the link line) will be used to compile the instantiated templates.

2.4.4 Compiling for 64–Bit Memory Model

Use the new -m64 option to specify the memory model of the target compilation. The resulting executable will work only on 64-bit UltraSPARC or x86 processors under Solaris OS or Linux OS running a 64-bit kernel. Compilation linking, and execution of 64-bit objects can only take place in a Solaris or Linux OS that supports 64-bit execution.
2.4.5 Diagnosing the Compiler

The -V option displays the name and version number of each program invoked by CC. The -v option displays the full command lines invoked by CC.

The --verbose=%all displays additional information about the compiler.

Any arguments on the command line that the compiler does not recognize are interpreted as linker options, object program file names, or library names.

The basic distinctions are:

- Unrecognized options, which are preceded by a dash (-) or a plus sign (+), generate warnings.
- Unrecognized nonoptions, which are not preceded by a dash or a plus sign, generate no warnings. (However, they are passed to the linker. If the linker does not recognize them, they generate linker error messages.)

In the following example, note that -bit is not recognized by CC and the option is passed on to the linker (ld), which tries to interpret it. Because single letter ld options can be strung together, the linker sees -bit as -b -i -t, all of which are legitimate ld options. This might not be what you intend or expect:

```
example% CC -bit move.cc
```

< -bit is not a recognized CC option

CC: Warning: Option -bit passed to ld, if ld is invoked, ignored otherwise

In the next example, the user intended to type the CC option -fast but omitted the leading dash. The compiler again passes the argument to the linker, which in turn interprets it as a file name:

```
example% CC fast move.cc
```

< The user meant to type -fast

move.CC:
ld: fatal: file fast: cannot open file; errno=2
ld: fatal: File processing errors. No output written to a.out

2.4.6 Understanding the Compiler Organization

The C++ compiler package consists of a front end, optimizer, code generator, assembler, template prelinker, and link editor. The CC command invokes each of these components automatically unless you use command-line options to specify otherwise.

Because any of these components may generate an error, and the components perform different tasks, it may be helpful to identify the component that generates an error. Use the -v and -dryrun options to help with this.
As shown in the following table, input files to the various compiler components have different file name suffixes. The suffix establishes the kind of compilation that is done. Refer to Table 2–1 for the meanings of the file suffixes.

**TABLE 2–2  Components of the C++ Compilation System**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Notes on Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccfe</td>
<td>Front end (compiler preprocessor and compiler)</td>
<td></td>
</tr>
<tr>
<td>iropt</td>
<td>Code optimizer</td>
<td><code>-x0[2-5], -fast</code></td>
</tr>
<tr>
<td>ir2hf</td>
<td>x86: Intermediate language translator</td>
<td><code>-x0[2-5], -fast</code></td>
</tr>
<tr>
<td>inline</td>
<td>SPARC: Inline expansion of assembly language templates</td>
<td><code>.ll</code> file specified</td>
</tr>
<tr>
<td>fbe</td>
<td>Assembler</td>
<td></td>
</tr>
<tr>
<td>cg</td>
<td>SPARC: Code generator, inliner, assembler</td>
<td></td>
</tr>
<tr>
<td>ube</td>
<td>x86: Code generator</td>
<td><code>-x0[2-5], -fast</code></td>
</tr>
<tr>
<td>CClink</td>
<td>Template pre-linker</td>
<td>used only with the <code>-instances=extern</code> option</td>
</tr>
<tr>
<td>ld</td>
<td>link editor</td>
<td></td>
</tr>
</tbody>
</table>

### 2.5 Preprocessing Directives and Names

This section discusses information about preprocessing directives that is specific to the C++ compiler.

#### 2.5.1 Pragmas

The preprocessor directive `pragma` is part of the C++ standard, but the form, content, and meaning of pragmas is different for every compiler. See Appendix B, "Pragmas," for details on the pragmas that the C++ compiler recognizes.

Sun C++ also supports the C99 keyword `_Pragma`. The two invocations

```c
#pragma dumpmacros(defs)
Pragma("dumpmacros(defs)")
```

are equivalent. To use `_Pragma` instead of `#pragma`, write the pragma text as a literal string, enclosed in parentheses as the one argument of the `_Pragma` keyword.
2.5.2 Macros With a Variable Number of Arguments

The C++ compiler accepts `#define` preprocessor directives of the following form.

```
#define identifier (...) replacement_list
#define identifier (identifier_list, ...) replacement_list
```

If the macro parameter list ends with an ellipsis, an invocation of the macro is allowed to have more arguments than there are macro parameters. The additional arguments are collected into a single string, including commas, that can be referenced by the name `__VA_ARGS__` in the macro replacement list. The following example demonstrates how to use a variable-argument-list macro.

```
#define debug(...) fprintf(stderr, __VA_ARGS__)
#define showlist(...) puts(#__VA_ARGS__)
#define report(test, ...) ((test)?puts(#test):
                                printf(__VA_ARGS__))

debug("Flag");
dump("X = %d\n", x);
showlist(The first, second, and third items.);
report(x>y, "x is %d but y is %d", x, y);
```

which results in the following:

```
fprintf(stderr, "Flag");
fprintf(stderr, "X = %d\n", x);
puts("The first, second, and third items.");
((x>y)?puts("x>y"):printf("x is %d but y is %d", x, y));
```

2.5.3 Predefined Names

Table A–2 in the appendix shows the predefined macros. You can use these values in such preprocessor conditionals as `#ifdef`. The `+p` option prevents the automatic definition of the `sun`, `unix`, `sparc`, and `i386` predefined macros.

2.5.4 `#error`

The `#error` directive no longer continues compilation after issuing a warning. The previous behavior of the directive was to issue a warning and continue compilation. The new behavior, consistent with other compilers, is to issue an error message and immediately halt compilation. The compiler quits and reports the failure.
2.6 Memory Requirements

The amount of memory a compilation requires depends on several parameters, including:

- Size of each procedure
- Level of optimization
- Limits set for virtual memory
- Size of the disk swap file

On the SPARC platform, if the optimizer runs out of memory, it tries to recover by retrying the current procedure at a lower level of optimization. The optimizer then resumes subsequent routines at the original level specified in the `-x0level` option on the command line.

If you compile a single source file that contains many routines, the compiler might run out of memory or swap space. Try reducing the level of optimization. Alternately, split the largest procedures into separate files of their own.

2.6.1 Swap Space Size

The `swap -s` command displays available swap space. See the `swap(1M)` man page for more information.

The following example demonstrates the use of the `swap` command:

```
example% swap -s
total: 40236k bytes allocated + 7280k reserved = 47516k used, 1058708k available
```

2.6.2 Increasing Swap Space

Use `mkfile(1M)` and `swap(1M)` to increase the size of the swap space on a workstation. (You must become superuser to do this.) The `mkfile` command creates a file of a specific size, and `swap -a` adds the file to the system swap space:

```
example# mkfile -v 90m /home/swapfile
/home/swapfile 94317840 bytes
example# /usr/sbin/swap -a /home/swapfile
```

2.6.3 Control of Virtual Memory

Compiling very large routines (thousands of lines of code in a single procedure) at `-x03` or higher can require a large amount of memory. In such cases, performance of the system might degrade. You can control this by limiting the amount of virtual memory available to a single process.
To limit virtual memory in an sh shell, use the ulimit command. See the sh(1) man page for more information.

The following example shows how to limit virtual memory to 4 Gbytes:

```
example$ ulimit -d 4000000
```

In a csh shell, use the limit command to limit virtual memory. See the csh(1) man page for more information.

The next example also shows how to limit virtual memory to 4 Gbytes:

```
example% limit datasize 4G
```

Each of these examples causes the optimizer to try to recover at 4 Gbytes of data space.

The limit on virtual memory cannot be greater than the system's total available swap space and, in practice, must be small enough to permit normal use of the system while a large compilation is in progress.

Be sure that no compilation consumes more than half the swap space.

With 8 Gbytes of swap space, use the following commands:

In an sh shell:

```
example$ ulimit -d 4000000
```

In a csh shell:

```
example% limit datasize 4G
```

The best setting depends on the degree of optimization requested and the amount of real memory and virtual memory available.

### 2.6.4 Memory Requirements

A workstation should have at least 2 gigabytes of memory; 1 gigabyte is recommended. See the product release readme (http://developers.sun.com/sunstudio/documentation/) for detailed requirements.
2.7 Using the strip Command with C++ Objects

The Unix strip command should not be used with C++ object files, as it can render those object files unusable.

2.8 Simplifying Commands

You can simplify complicated compiler commands by defining special shell aliases, using the CCFLAGS environment variable, or by using make.

2.8.1 Using Aliases Within the C Shell

The following example defines an alias for a command with frequently used options.

```
example% alias CCfx "CC -fast -xnolibmil"
```

The next example uses the alias CCfx.

```
example% CCfx any.C
```

The command CCfx is now the same as:

```
example% CC -fast -xnolibmil any.C
```

2.8.2 Using CCFLAGS to Specify Compile Options

You can specify options by setting the CCFLAGS variable.

The CCFLAGS variable can be used explicitly in the command line. The following example shows how to set CCFLAGS (C Shell):

```
example% setenv CCFLAGS '-xO2 -m64'
```

The next example uses CCFLAGS explicitly.

```
example% CC $CCFLAGS any.cc
```

When you use make, if the CCFLAGS variable is set as in the preceding example and the makefile's compilation rules are implicit, then invoking make will result in a compilation equivalent to:

```
CC -xO2 -m64 files...
```
2.8.3 Using make

The make utility is a very powerful program development tool that you can easily use with all Sun compilers. See the make(1S) man page for additional information.

2.8.3.1 Using CCFLAGS Within make

When you are using the *implicit* compilation rules of the makefile (that is, there is no C++ compile line), the make program uses CCFLAGS automatically.
This chapter explains how to use the command-line C++ compiler options and then summarizes their use by function. Detailed explanations of the options are provided in “A.2 Option Reference” on page 212.

3.1 Syntax

The following table shows examples of typical option syntax formats that are used in this book.

<table>
<thead>
<tr>
<th>Syntax Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-option</td>
<td>-E</td>
</tr>
<tr>
<td>-option=value</td>
<td>-I pathname</td>
</tr>
<tr>
<td>-option=value</td>
<td>-xunroll=4</td>
</tr>
<tr>
<td>-option value</td>
<td>-o filename</td>
</tr>
</tbody>
</table>

Parentheses, braces, brackets, pipe characters, and ellipses are *metacharacters* used in the descriptions of the options and are not part of the options themselves. See the typographical conventions in the Preface to this manual for a detailed explanation of the usage syntax.
3.2 General Guidelines

Some general guidelines for the C++ compiler options are:

- The -l option links with library libc.lib.a (or libc.lib.so). It is always safer to put -l after the source and object files to ensure the order in which libraries are searched.

- In general, processing of the compiler options is from left to right (with the exception that -U options are processed after all -D options), allowing selective overriding of macro options (options that include other options). This rule does not apply to linker options.


- The -D option accumulates; however, multiple -D options for the same name override each other.

Source files, object files, and libraries are compiled and linked in the order in which they appear on the command line.

3.3 Options Summarized by Function

In this section, the compiler options are grouped by function to provide a quick reference. For a detailed description of each option, refer to Appendix A, “C++ Compiler Options.”

The options apply to all platforms except as noted; features that are unique to the Solaris OS on SPARC-based systems are identified as SPARC, and the features that are unique to the Solaris OS on x86-based systems are identified as x86.

3.3.1 Code Generation Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-compat</td>
<td>Sets the major release compatibility mode of the compiler.</td>
</tr>
<tr>
<td>+e{0</td>
<td>1}</td>
</tr>
<tr>
<td>-g</td>
<td>Compiles for use with the debugger.</td>
</tr>
<tr>
<td>-KPIC</td>
<td>Produces position-independent code.</td>
</tr>
<tr>
<td>-Kpic</td>
<td>Produces position-independent code.</td>
</tr>
<tr>
<td>-mt</td>
<td>Compiles and links for multithreaded code.</td>
</tr>
</tbody>
</table>
### 3.3 Options Summarized by Function

#### 3.3.2 Compile-Time Performance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>--instlib</td>
<td>Inhibits the generation of template instances that are already present in the designated library.</td>
</tr>
<tr>
<td>-m32</td>
<td>-m64</td>
</tr>
<tr>
<td>-xinstrument</td>
<td>Compiles and instruments your program for analysis by the Thread Analyzer.</td>
</tr>
<tr>
<td>-xjobs</td>
<td>Sets the number of processes the compiler can create to complete its work.</td>
</tr>
<tr>
<td>-xpch</td>
<td>May reduce compile time for applications whose source files share a common set of include files.</td>
</tr>
<tr>
<td>-xpchstop</td>
<td>Specifies the last include file to be considered in creating a pre-compiled header file with -xpch.</td>
</tr>
</tbody>
</table>
### 3.3.3 Compile-Time and Link-Time Options

The following table lists the options that must be specified both at link-time and at compile-time.

**TABLE 3-4  Compile-Time and Link-Time Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fast</td>
<td>Selects the optimum combination of compilation options for speed of executable code.</td>
</tr>
<tr>
<td>-m32/-m64</td>
<td>Specifies the memory model for the compiled binary object.</td>
</tr>
<tr>
<td>-mt</td>
<td>Macro option that expands to -D_REENTRANT -lthread.</td>
</tr>
<tr>
<td>-xarch</td>
<td>Specify instruction set architecture.</td>
</tr>
<tr>
<td>-xautopar</td>
<td>Turns on automatic parallelization for multiple processors.</td>
</tr>
<tr>
<td>-xhwcprof</td>
<td>(SPARC) Enables compiler support for hardware counter-based profiling.</td>
</tr>
<tr>
<td>-xipo</td>
<td>Performs whole-program optimizations by invoking an interprocedural analysis component.</td>
</tr>
<tr>
<td>-xlinkopt</td>
<td>Performs link-time optimizations on relocatable object files.</td>
</tr>
<tr>
<td>-xmemalign</td>
<td>(SPARC) Specify maximum assumed memory alignment and behavior of misaligned data accesses.</td>
</tr>
<tr>
<td>-xopenmp</td>
<td>Supports the OpenMP interface for explicit parallelization including a set of source code directives, run-time library routines, and environment variables.</td>
</tr>
<tr>
<td>-xpagesize</td>
<td>Sets the preferred page size for the stack and the heap.</td>
</tr>
<tr>
<td>-xpagesize_heap</td>
<td>Sets the preferred page size for the heap.</td>
</tr>
</tbody>
</table>
### 3.3.4 Debugging Options

#### TABLE 3–5  Debugging Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>+d</td>
<td>Does not expand C++ inline functions.</td>
</tr>
<tr>
<td>--dryrun</td>
<td>Shows options passed by the driver to the compiler, but does not compile.</td>
</tr>
<tr>
<td>–E</td>
<td>Runs only the preprocessor on the C++ source files and sends result to stdout. Does not compile.</td>
</tr>
<tr>
<td>–g</td>
<td>Compiles for use with the debugger.</td>
</tr>
<tr>
<td>–g0</td>
<td>Compiles for debugging, but doesn’t disable inlining.</td>
</tr>
<tr>
<td>–H</td>
<td>Prints path names of included files.</td>
</tr>
<tr>
<td>–keeptmp</td>
<td>Retains temporary files created during compilation.</td>
</tr>
<tr>
<td>–migration</td>
<td>Explains where to get information about migrating from earlier compilers.</td>
</tr>
<tr>
<td>–P</td>
<td>Only preprocesses source; outputs to .i file.</td>
</tr>
<tr>
<td>–Qoption</td>
<td>Passes an option directly to a compilation phase.</td>
</tr>
<tr>
<td>–readme</td>
<td>Displays the content of the online README file.</td>
</tr>
<tr>
<td>–s</td>
<td>Strips the symbol table out of the executable file, thus preventing the ability to debug code.</td>
</tr>
<tr>
<td>–temp=dir</td>
<td>Defines directory for temporary files.</td>
</tr>
<tr>
<td>–verbose=vlst</td>
<td>Controls compiler verbosity.</td>
</tr>
<tr>
<td>–xcheck</td>
<td>Adds a runtime check for stack overflow.</td>
</tr>
</tbody>
</table>
3.3.5 Floating-Point Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fma</td>
<td>(SPARC) Enables automatic generation of floating-point, fused, multiply-add instructions.</td>
</tr>
<tr>
<td>-fns[=no</td>
<td>yes]</td>
</tr>
<tr>
<td>-fprecision=p</td>
<td>x86: Sets floating-point precision mode.</td>
</tr>
<tr>
<td>-fround=r</td>
<td>Sets IEEE rounding mode in effect at startup.</td>
</tr>
<tr>
<td>-fsimple=n</td>
<td>Sets floating-point optimization preferences.</td>
</tr>
<tr>
<td>-fstore=x86</td>
<td>Forcibly forces precision of floating-point expressions.</td>
</tr>
<tr>
<td>-ftrap=lst</td>
<td>Sets IEEE trapping mode in effect at startup.</td>
</tr>
<tr>
<td>-nofstore=x86</td>
<td>x86: Disables forced precision of expression.</td>
</tr>
<tr>
<td>-xlibmieee</td>
<td>Causes libm to return IEEE 754 values for math routines in exceptional cases.</td>
</tr>
</tbody>
</table>

3.3.6 Language Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-compat</td>
<td>Sets the major release compatibility mode of the compiler.</td>
</tr>
<tr>
<td>-features=</td>
<td>Enables or disables various C++ language features.</td>
</tr>
</tbody>
</table>
### 3.3.7 Library Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-xbinding</code></td>
<td>Requests symbolic, dynamic, or static library linking.</td>
</tr>
<tr>
<td><code>-dy{n}</code></td>
<td>Allows or disallows dynamic libraries for the entire executable.</td>
</tr>
<tr>
<td><code>-G</code></td>
<td>Builds a dynamic shared library instead of an executable file.</td>
</tr>
<tr>
<td><code>-h name</code></td>
<td>Assigns an internal name to the generated dynamic shared library.</td>
</tr>
<tr>
<td><code>-i</code></td>
<td>Tells <code>ld(1)</code> to ignore any <code>LD_LIBRARY_PATH</code> setting.</td>
</tr>
<tr>
<td><code>-ldir</code></td>
<td>Adds <code>dir</code> to the list of directories to be searched for libraries.</td>
</tr>
<tr>
<td><code>-lib</code></td>
<td>Adds <code>liblib.a</code> or <code>liblib.so</code> to the linker’s library search list.</td>
</tr>
<tr>
<td><code>-library=llst</code></td>
<td>Forces inclusion of specific libraries and associated files into compilation and linking.</td>
</tr>
<tr>
<td><code>-mt</code></td>
<td>Compiles and links for multithreaded code.</td>
</tr>
<tr>
<td><code>-norunpath</code></td>
<td>Does not build path for libraries into executable.</td>
</tr>
<tr>
<td><code>-Rplst</code></td>
<td>Builds dynamic library search paths into the executable file.</td>
</tr>
<tr>
<td><code>-staticlib=llst</code></td>
<td>Indicates which C++ libraries are to be linked statically.</td>
</tr>
<tr>
<td><code>-xar</code></td>
<td>Creates archive libraries.</td>
</tr>
<tr>
<td><code>-xbuiltin[=opt]</code></td>
<td>Enables or disables better optimization of standard library calls</td>
</tr>
<tr>
<td><code>-xia</code></td>
<td>(Solaris) Links the appropriate interval arithmetic libraries and sets a suitable floating-point environment.</td>
</tr>
</tbody>
</table>
### TABLE 3-8 Library Options (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-xlang=ll</td>
<td>Includes the appropriate runtime libraries and ensures the proper runtime environment for the specified language.</td>
</tr>
<tr>
<td>-xlibmeee</td>
<td>Causes libm to return IEEE 754 values for math routines in exceptional cases.</td>
</tr>
<tr>
<td>-xlibmil</td>
<td>Inlines selected libm library routines for optimization.</td>
</tr>
<tr>
<td>-xlibmopt</td>
<td>Uses library of optimized math routines.</td>
</tr>
<tr>
<td>-xnolib</td>
<td>Disables linking with default system libraries.</td>
</tr>
<tr>
<td>-xnolibmil</td>
<td>Cancels -xlibmil on the command line.</td>
</tr>
<tr>
<td>-xnolibmopt</td>
<td>Does not use the math routine library.</td>
</tr>
</tbody>
</table>

### 3.3.8 Obsolete Options

Note – The following options are either currently obsolete and so no longer accepted by the compiler, or they will be removed in a future release.

### TABLE 3-9 Obsolete Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-library=%all</td>
<td>Obsolete option that will be removed in a future release.</td>
</tr>
<tr>
<td>-xlic_lib=sunperf</td>
<td>Use -library=sunperf to link to the Sun Performance Library.</td>
</tr>
<tr>
<td>-xlicinfo</td>
<td>Deprecated.</td>
</tr>
<tr>
<td>-noqueue</td>
<td>Disables license queueing.</td>
</tr>
<tr>
<td>-ptr</td>
<td>Ignored by the compiler. A future release of the compiler may reuse this option using a different behavior.</td>
</tr>
<tr>
<td>-sb, -sbfast, -xsb,</td>
<td>Deprecated and silently ignored.</td>
</tr>
<tr>
<td>-xsbfast</td>
<td></td>
</tr>
<tr>
<td>-vdelx</td>
<td>Obsolete option that will be removed in a future release.</td>
</tr>
<tr>
<td>-x386</td>
<td>Use an appropriate -xtarget option.</td>
</tr>
<tr>
<td>-x486</td>
<td>Use an appropriate -xtarget option.</td>
</tr>
<tr>
<td>-xcg89</td>
<td>Use -xtarget=ss2</td>
</tr>
<tr>
<td>-xcrossfile</td>
<td>Use -xipo instead.</td>
</tr>
</tbody>
</table>
### 3.3.9 Output Options

#### TABLE 3–10  Output Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>Compiles only; produces object (.o) files, but suppresses linking.</td>
</tr>
<tr>
<td>--dryrun</td>
<td>Shows all the command lines issued by the driver to the compiler, but does not compile.</td>
</tr>
<tr>
<td>-E</td>
<td>Runs only the preprocessor on the C++ source files and sends result to stdout. Does not compile.</td>
</tr>
<tr>
<td>-erroff</td>
<td>Suppresses compiler warning messages.</td>
</tr>
<tr>
<td>-errtags</td>
<td>Displays the message tag for each warning message.</td>
</tr>
<tr>
<td>-errwarn</td>
<td>If the indicated warning message is issued, cc exits with a failure status.</td>
</tr>
<tr>
<td>-filt</td>
<td>Suppresses the filtering that the compiler applies to linker error messages.</td>
</tr>
<tr>
<td>-G</td>
<td>Builds a dynamic shared library instead of an executable file.</td>
</tr>
<tr>
<td>-H</td>
<td>Prints path names of included files.</td>
</tr>
<tr>
<td>-migration</td>
<td>Explains where to get information about migrating from earlier compilers.</td>
</tr>
<tr>
<td>-o filename</td>
<td>Sets name of the output or executable file to filename.</td>
</tr>
<tr>
<td>-P</td>
<td>Only preprocesses source; outputs to .i file.</td>
</tr>
<tr>
<td>-Qproduce sourcetype</td>
<td>Causes the CC driver to produce output of the type sourcetype.</td>
</tr>
<tr>
<td>-s</td>
<td>Strips the symbol table out of the executable file.</td>
</tr>
<tr>
<td>-verbose=vlst</td>
<td>Controls compiler verbosity.</td>
</tr>
<tr>
<td>Option</td>
<td>Action</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>+w</td>
<td>Prints extra warnings where necessary.</td>
</tr>
<tr>
<td>+w2</td>
<td>Print still more warnings where appropriate.</td>
</tr>
<tr>
<td>-w</td>
<td>Suppresses warning messages.</td>
</tr>
<tr>
<td>-xdumpmacros</td>
<td>Prints information about macros such as definition, location defined and undefined, and locations used.</td>
</tr>
<tr>
<td>-xe</td>
<td>Performs only syntax and semantic checking on the source file, but does not produce any object or executable code.</td>
</tr>
<tr>
<td>-xhelp=flags</td>
<td>Displays a summary list of compiler options</td>
</tr>
<tr>
<td>-xhelp=readme</td>
<td>Displays the contents of the online README file.</td>
</tr>
<tr>
<td>-xM</td>
<td>Outputs makefile dependency information.</td>
</tr>
<tr>
<td>-xM1</td>
<td>Generates dependency information, but excludes /usr/include.</td>
</tr>
<tr>
<td>-xtime</td>
<td>Reports execution time for each compilation phase.</td>
</tr>
<tr>
<td>-xwe</td>
<td>Converts all warnings to errors.</td>
</tr>
<tr>
<td>-z arg</td>
<td>Linker option.</td>
</tr>
</tbody>
</table>

### 3.3.10 Run-Time Performance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fast</td>
<td>Selects a combination of compilation options for optimum execution speed for some programs.</td>
</tr>
<tr>
<td>-fma</td>
<td>(SPARC) Enables automatic generation of floating-point, fused, multiply-add instructions.</td>
</tr>
<tr>
<td>-g</td>
<td>Instructs both the compiler and the linker to prepare the program for performance analysis (and for debugging).</td>
</tr>
<tr>
<td>-s</td>
<td>Strips the symbol table out of the executable.</td>
</tr>
<tr>
<td>-m32</td>
<td>-m64</td>
</tr>
<tr>
<td>-xalias_level</td>
<td>Enables the compiler to perform type-based alias analysis and optimizations.</td>
</tr>
<tr>
<td>Option</td>
<td>Action</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-xarch=isa</td>
<td>Specifies target architecture instruction set.</td>
</tr>
<tr>
<td>-xbinopt</td>
<td>Prepares the binary for later optimizations, transformations and analysis.</td>
</tr>
<tr>
<td>-xbuiltin[=opt]</td>
<td>Enables or disables better optimization of standard library calls</td>
</tr>
<tr>
<td>-xcache=c</td>
<td>(SPARC) Defines target cache properties for the optimizer.</td>
</tr>
<tr>
<td>-xcg89</td>
<td>Compiles for generic SPARC v7 architecture.</td>
</tr>
<tr>
<td>-xcg92</td>
<td>Compiles for SPARC V8 architecture.</td>
</tr>
<tr>
<td>-xchip=c</td>
<td>Specifies target processor chip.</td>
</tr>
<tr>
<td>-xF</td>
<td>Enables linker reordering of functions and variables.</td>
</tr>
<tr>
<td>-xinline=flst</td>
<td>Specifies which user-written routines can be inlined by the optimizer.</td>
</tr>
<tr>
<td>-xipo</td>
<td>Performs interprocedural optimizations.</td>
</tr>
<tr>
<td>-xlibmil</td>
<td>Inlines selected <code>libm</code> library routines for optimization.</td>
</tr>
<tr>
<td>-xlibmopt</td>
<td>Uses a library of optimized math routines.</td>
</tr>
<tr>
<td>-xlinkopt</td>
<td>(SPARC) Performs link-time optimization on the resulting executable or dynamic library over and above any optimizations in the object files.</td>
</tr>
<tr>
<td>-xmemalign=ab</td>
<td>(SPARC) Specify maximum assumed memory alignment and behavior of misaligned data accesses.</td>
</tr>
<tr>
<td>-xnolibmil</td>
<td>Cancels <code>-xlibmil</code> on the command line.</td>
</tr>
<tr>
<td>-xnolibmopt</td>
<td>Does not use the math routine library.</td>
</tr>
<tr>
<td>-xlevel</td>
<td>Specifies optimization level to level.</td>
</tr>
<tr>
<td>-xpysize</td>
<td>Sets the preferred page size for the stack and the heap.</td>
</tr>
<tr>
<td>-xpysize_heap</td>
<td>Sets the preferred page size for the heap.</td>
</tr>
<tr>
<td>-xpysize_stack</td>
<td>Sets the preferred page size for the stack.</td>
</tr>
<tr>
<td>-xprefetch[=lst]</td>
<td>Enables prefetch instructions on architectures that support prefetch.</td>
</tr>
<tr>
<td>-xprefetch_level</td>
<td>Control the aggressiveness of automatic insertion of prefetch instructions as set by <code>-xprefetch=auto</code></td>
</tr>
<tr>
<td>-xprofile</td>
<td>Collects or optimizes using runtime profiling data.</td>
</tr>
<tr>
<td>-xregs=rlst</td>
<td>Controls scratch register use.</td>
</tr>
<tr>
<td>-xsafe=mem</td>
<td>(SPARC) Allows no memory-based traps.</td>
</tr>
</tbody>
</table>
### 3.3.11 Preprocessor Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-D name</code></td>
<td>Defines symbol <code>name</code> to the preprocessor.</td>
</tr>
<tr>
<td><code>-E</code></td>
<td>Runs only the preprocessor on the C++ source files and sends result to <code>stdout</code>. Does not compile.</td>
</tr>
<tr>
<td><code>-H</code></td>
<td>Prints path names of included files.</td>
</tr>
<tr>
<td><code>-P</code></td>
<td>Only preprocesses source; outputs to <code>.i</code> file.</td>
</tr>
<tr>
<td><code>-U name</code></td>
<td>Deletes initial definition of preprocessor symbol <code>name</code>.</td>
</tr>
<tr>
<td><code>-xM</code></td>
<td>Outputs makefile dependency information.</td>
</tr>
<tr>
<td><code>-xM1</code></td>
<td>Generates dependency information, but excludes <code>/usr/include</code>.</td>
</tr>
</tbody>
</table>

### 3.3.12 Profiling Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-p</code></td>
<td>Prepares the object code to collect data for profiling using <code>prof</code>.</td>
</tr>
<tr>
<td><code>-xa</code></td>
<td>Generates code for profiling.</td>
</tr>
<tr>
<td><code>-xpg</code></td>
<td>Compiles for profiling with the <code>gprof</code> profiler.</td>
</tr>
<tr>
<td><code>-xprofile</code></td>
<td>Collects or optimizes using runtime profiling data.</td>
</tr>
</tbody>
</table>
3.3.13 Reference Options

**TABLE 3–14** Reference Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>--migration</td>
<td>Explains where to get information about migrating from earlier compilers.</td>
</tr>
<tr>
<td>--xhelp=flags</td>
<td>Displays a summary list of compiler options.</td>
</tr>
<tr>
<td>--xhelp=readme</td>
<td>Displays the contents of the online README file.</td>
</tr>
</tbody>
</table>

3.3.14 Source Options

**TABLE 3–15** Source Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-H</td>
<td>Prints path names of included files.</td>
</tr>
<tr>
<td>-I pathname</td>
<td>Adds pathname to the include file search path.</td>
</tr>
<tr>
<td>-I-</td>
<td>Changes the include-file search rules</td>
</tr>
<tr>
<td>-xM</td>
<td>Outputs makefile dependency information.</td>
</tr>
<tr>
<td>-xM1</td>
<td>Generates dependency information, but excludes /usr/include.</td>
</tr>
</tbody>
</table>

3.3.15 Template Options

**TABLE 3–16** Template Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-instances=a</td>
<td>Controls the placement and linkage of template instances.</td>
</tr>
<tr>
<td>-template=wlist</td>
<td>Enables or disables various template options.</td>
</tr>
</tbody>
</table>
### Thread Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-mt</td>
<td>Compiles and links for multithreaded code.</td>
</tr>
<tr>
<td>-xsafe=mem</td>
<td>(SPARC) Allows no memory-based traps.</td>
</tr>
<tr>
<td>-xthreadvar</td>
<td>(SPARC) Changes the default thread-local storage access mode.</td>
</tr>
</tbody>
</table>
PART II

Writing C++ Programs
This chapter documents the language extensions specific to this compiler. The compiler does not recognize some of the features described in this chapter unless you specify certain compiler options on the command line. The relevant compiler options are listed in each section as appropriate.

The `-features=extensions` option enables you to compile nonstandard code that is commonly accepted by other C++ compilers. You can use this option when you must compile invalid code and you are not permitted to modify the code to make it valid.

This chapter describes the language extensions that the compiler supports when you use the `-features=extensions` options.

**Note** – You can easily turn each supported instance of invalid code into valid code that all compilers will accept. If you are allowed to make the code valid, you should do so instead of using this option. Using the `-features=extensions` option perpetuates invalid code that will be rejected by some compilers.

## 4.1 Linker Scoping

Use the following declaration specifiers to help constrain declarations and definitions of extern symbols. The scoping restraints you specify for a static archive or an object file will not take effect until the file is linked into a shared library or an executable. Despite this, the compiler can still perform some optimization given the presence of the linker scoping specifiers.

By using these specifiers, you no longer need to use mapfiles for linker scoping. You can also control the default setting for variable scoping by specifying `-xldscope` on the command line.

For more information, see "A.2.136 -xldscope=[v]" on page 310.
TABLE 4–1 Linker Scoping Declaration Specifiers

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>__global</td>
<td>Symbol definitions have global linker scoping and is the least restrictive linker scoping. All references to the symbol bind to the definition in the first dynamic load module that defines the symbol. This linker scoping is the current linker scoping for extern symbols.</td>
</tr>
<tr>
<td>__symbolic</td>
<td>Symbol definitions have symbolic linker scoping and is more restrictive than global linker scoping. All references to the symbol from within the dynamic load module being linked bind to the symbol defined within the module. Outside of the module, the symbol appears as though it were global. This linker scoping corresponds to the linker option -Bsymbolic. Although you cannot use -Bsymbolic with C++ libraries, you can use the __symbolic specifier without causing problems. See ld(1) for more information on the linker.</td>
</tr>
<tr>
<td>__hidden</td>
<td>Symbol definitions have hidden linker scoping. Hidden linker scoping is more restrictive than symbolic and global linker scoping. All references within a dynamic load module bind to a definition within that module. The symbol will not be visible outside of the module.</td>
</tr>
</tbody>
</table>

A symbol definition may be redeclared with a more restrictive specifier, but may not be redeclared with a less restrictive specifier. A symbol may not be declared with a different specifier once the symbol has been defined.

__global is the least restrictive scoping, __symbolic is more restrictive, and __hidden is the most restrictive scoping.

All virtual functions must be visible to all compilation units that include the class definition because the declaration of virtual functions affects the construction and interpretation of virtual tables.

You can apply the linker scoping specifiers to struct, class, and union declarations and definitions because C++ classes may require generation of implicit information, such as virtual tables and run-time type information. The specifier, in this case, follows the struct, class, or union keyword. Such an application implies the same linker scoping for all its implicit members.

4.1.1 Compatibility with Microsoft Windows

For compatibility with similar scoping features in Microsoft Visual C++ (MSVC++) for dynamic libraries, the following syntax is also supported:

__declspec(dllexport) is equivalent to __symbolic
4.2 Thread-LocalStorage

Take advantage of thread-local storage by declaring thread-local variables. A thread-local variable declaration consists of a normal variable declaration with the addition of the declaration specifier __thread. For more information, see "A.2.182 -xthreadvar [=o]" on page 356.

You must include the __thread specifier in the first declaration of the thread variable. Variables that you declare with the __thread specifier are bound as they would be without the __thread specifier.

You can declare variables only of static duration with the __thread specifier. Variables with static duration include file global, file static, function local static, and class static member. You should not declare variables with dynamic or automatic duration with the __thread specifier. A thread variable can have a static initializer, but it cannot have a dynamic initializer or destructors. For example, __thread int x = 4; is permitted, but __thread int x = f(); is not. A thread variable should not have a type with non-trivial constructors and destructors. In particular, a thread variable may not have type std::string.

The address-of operator (&) for a thread variable is evaluated at run time and returns the address of the current thread's variable. Therefore, the address of a thread variable is not a constant.

The address of a thread variable is stable for the lifetime of the corresponding thread. Any thread in the process can freely use the address of a thread variable during the variable's lifetime. You cannot use a thread variable's address after its thread terminates. All addresses of a thread's variables are invalid after the thread's termination.
4.3 Overriding With Less Restrictive Virtual Functions

The C++ standard says that an overriding virtual function must not be less restrictive in the exceptions it allows than any function it overrides. It can have the same restrictions or be more restrictive. Note that the absence of an exception specification allows any exception.

Suppose, for example, that you call a function through a pointer to a base class. If the function has an exception specification, you can count on no other exceptions being thrown. If the overriding function has a less-restrictive specification, an unexpected exception could be thrown, which can result in bizarre program behavior followed by a program abort. This is the reason for the rule.

When you use -features=extensions, the compiler will allow overriding functions with less-restrictive exception specifications.

4.4 Making Forward Declarations of enum Types and Variables

When you use -features=extensions, the compiler allows the forward declaration of enum types and variables. In addition, the compiler allows the declaration of a variable with an incomplete enum type. The compiler will always assume an incomplete enum type to have the same size and range as type int on the current platform.

The following two lines show an example of invalid code that will compile when you use the -features=extensions option.

```c
enum E; // invalid: forward declaration of enum not allowed
E e;   // invalid: type E is incomplete
```

Because enum definitions cannot reference one another, and no enum definition can cross-reference another type, the forward declaration of an enumeration type is never necessary. To make the code valid, you can always provide the full definition of the enum before it is used.

**Note** – On 64-bit architectures, it is possible for an enum to require a size that is larger than type int. If that is the case, and if the forward declaration and the definition are visible in the same compilation, the compiler will emit an error. If the actual size is not the assumed size and the compiler does not see the discrepancy, the code will compile and link, but might not run properly. Mysterious program behavior can occur, particularly if an 8-byte value is stored in a 4-byte variable.
4.5 **Using Incomplete enum Types**

When you use `-features=extensions`, incomplete enum types are taken as forward declarations. For example, the following invalid code will compile when you use the `-features=extensions` option.

```cpp
typedef enum E F; // invalid, E is incomplete
```

As noted previously, you can always include the definition of an enum type before it is used.

4.6 **Using an enum Name as a Scope Qualifier**

Because an enum declaration does not introduce a scope, an enum name cannot be used as a scope qualifier. For example, the following code is invalid.

```cpp
enum E {e1, e2, e3};
int i = E::e1; // invalid: E is not a scope name
```

To compile this invalid code, use the `-features=extensions` option. The `-features=extensions` option instructs the compiler to ignore a scope qualifier if it is the name of an enum type.

To make the code valid, remove the invalid qualifier `E::`.

---

**Note** – Use of this option increases the possibility of typographical errors yielding incorrect programs that compile without error messages.

4.7 **Using Anonymous struct Declarations**

An anonymous struct declaration is a declaration that declares neither a tag for the struct, nor an object or typedef name. Anonymous structs are not allowed in C++.

The `-features=extensions` option allows the use of an anonymous `struct` declaration, but only as member of a union.

The following code is an example of an invalid anonymous `struct` declaration that compiles when you use the `-features=extensions` option.

```cpp
union U {
    struct {
        int a;
        double b;
    }; // invalid: anonymous struct
```
struct {
    char* c;
    unsigned d;
}; // invalid: anonymous struct

The names of the struct members are visible without qualification by a struct member name. Given the definition of U in this code example, you can write:

U u;
    u.a = 1;

Anonymous structs are subject to the same limitations as anonymous unions.

Note that you can make the code valid by giving a name to each struct, such as:

union U {
    struct {
        int a;
        double b;
    } A;
    struct {
        char* c;
        unsigned d;
    } B;
};
    U u;
        U.A.a = 1;

4.8 Passing the Address of an Anonymous Class Instance

You are not allowed to take the address of a temporary variable. For example, the following code is invalid because it takes the address of a variable created by a constructor call. However, the compiler accepts this invalid code when you use the -features=extensions option.

class C {
public:
    C(int);
    ...
};
void f1(C*);
int main()
{
    f1(&C(2)); // invalid
}

Note that you can make this code valid by using an explicit variable.
C c(2);
f1(&c);

The temporary object is destroyed when the function returns. Ensuring that the address of the temporary variable is not retained is the programmer’s responsibility. In addition, the data that is stored in the temporary variable (for example, by f1) is lost when the temporary variable is destroyed.

4.9 Declaring a Static Namespace-Scope Function as a Class Friend

The following code is invalid.

class A {
    friend static void foo(<args>);
    ... 
};

Because a class name has external linkage and all definitions must be identical, friend functions must also have external linkage. However, when you use the -features=extensions option, the compiler accepts this code.

Presumably the programmer’s intent with this invalid code was to provide a nonmember "helper" function in the implementation file for class A. You can get the same effect by making foo a static member function. You can make it private if you do not want clients to call the function.

Note - If you use this extension, your class can be “hijacked” by any client. Any client can include the class header, then define its own static function foo, which will automatically be a friend of the class. The effect will be as if you made all members of the class public.

4.10 Using the Predefined __func__ Symbol for Function Name

When you use -features=extensions, the compiler implicitly declares the identifier __func__ in each function as a static array of const char. If the program uses the identifier, the compiler also provides the following definition where function-name is the unadorned name of the function. Class membership, namespaces, and overloading are not reflected in the name.

static const char __func__[] = "function-name";

For example, consider the following code fragment.
#include <stdio.h>
void myfunc(void)
{
    printf("%s\n", __func__); 
}

Each time the function is called, it will print the following to the standard output stream.

myfunc

4.11 The __packed__ Attribute

This attribute, attached to struct or union type definition, specifies that each member (other than zero-width bitfields) of the structure or union is placed to minimize the memory required. When attached to an enum definition, it indicates that the smallest integral type should be used.

Specifying this attribute for struct and union types is equivalent to specifying the packed attribute on each of the structure or union members.

In the following example struct my_packed_struct’s members are packed closely together, but the internal layout of its s member is not packed. To do that, struct my_unpacked_struct would also need to be packed.

```c
struct my_unpacked_struct
{
    char c;
    int i;
};

struct __attribute__((__packed__)) my_packed_struct
{
    char c;
    int i;
    struct my_unpacked_struct s;
};
```

You may only specify this attribute on the definition of a enum, struct or union, and not on a typedef that does not also define the enumerated type, structure or union.
The file organization of a C++ program requires more care than is typical for a C program. This chapter describes how to set up your header files and your template definitions.

5.1 Header Files

Creating an effective header file can be difficult. Often your header file must adapt to different versions of both C and C++. To accommodate templates, make sure your header file is tolerant of multiple inclusions (idempotent).

5.1.1 Language-Adaptable Header Files

You might need to develop header files for inclusion in both C and C++ programs. However, Kernighan and Ritchie C (K&R C), also known as "classic C," ANSI C, Annotated Reference Manual C++ (ARM C++), and ISO C++ sometimes require different declarations or definitions for the same program element within a single header file. (See the C++ Migration Guide for additional information on the variations between languages and versions.) To make header files acceptable to all these standards, you might need to use conditional compilation based on the existence or value of the preprocessor macros __STDC__ and __cplusplus.

The macro __STDC__ is not defined in K&R C, but is defined in both ANSI C and C++. Use this macro to separate K&R C code from ANSI C or C++ code. This macro is most useful for separating prototyped from nonprototyped function definitions.

```
#ifdef __STDC__
    int function(char*,...);  // C++ & ANSI C declaration
#else
    int function();          // K&R C
#endif
```

The macro __cplusplus is not defined in C, but is defined in C++. 

Note – Early versions of C++ defined the macro `c_plusplus` instead of `__cplusplus`. The macro `c_plusplus` is no longer defined.

Use the definition of the `__cplusplus` macro to separate C and C++. This macro is most useful in guarding the specification of an `extern "C"` interface for function declarations, as shown in the following example. To prevent inconsistent specification of `extern "C"`, never place an `#include` directive within the scope of an `extern "C"` linkage specification.

```c
#include "header.h"
...  // ... other include files...
#if defined(__cplusplus)
extern "C" {
#endif
int g1();
int g2();
int g3()
#if defined(__cplusplus)
}
#endif
```

In ARM C++, the `__cplusplus` macro has a value of 1. In ISO C++, the macro has the value 199711L (the year and month of the standard expressed as a long constant). Use the value of this macro to separate ARM C++ from ISO C++. The macro value is most useful for guarding changes in template syntax.

```c
// template function specialization
#if __cplusplus < 199711L
int power(int,int);  // ARM C++
#else
template <> int power(int,int);  // ISO C++
#endif
```

5.1.2 Idempotent Header Files

Your header files should be idempotent. That is, the effect of including a header file many times should be exactly the same as including the header file only once. This property is especially important for templates. You can best accomplish idempotency by setting preprocessor conditions that prevent the body of your header file from appearing more than once.

```c
#ifndef HEADER_H
#define HEADER_H
/* contents of header file */
#endif
```
5.2 Template Definitions

You can organize your template definitions in two ways: with definitions included and with definitions separated. The definitions-included organization allows greater control over template compilation.

5.2.1 Template Definitions Included

When you put the declarations and definitions for a template within the file that uses the template, the organization is definitions-included. For example:

main.cc

```cpp
template <class Number> Number twice(Number original);
template <class Number> Number twice(Number original )
    { return original + original; }
int main()
    { return twice<int>(-3); }
```

When a file using a template includes a file that contains both the template’s declaration and the template’s definition, the file that uses the template also has the definitions-included organization. For example:

twice.h

```cpp
#ifndef TWICE_H
#define TWICE_H
template <class Number> Number twice(Number original);
template <class Number> Number twice(Number original )
    { return original + original; }
#endif
```

main.cc

```cpp
#include “twice.h”
int main()
    { return twice(-3); }
```

**Note** – It is very important to make your template headers idempotent. (See “5.1.2 Idempotent Header Files” on page 72.)
5.2 Template Definitions Separate

Another way to organize template definitions is to keep the definitions in template definition files, as shown in the following example.

```
twice.h

#ifndef TWICE_H
#define TWICE_H

template <class Number>
Number twice(Number original);
#endif TWICE_H

twice.cc

template <class Number>
Number twice(Number original)
{
    return original + original;
}

main.cc

#include "twice.h"

int main()
{
    return twice<int>({ -3 });
}
```

Template definition files must not include any non-idempotent header files and often need not include any header files at all. (See "5.1.2 Idempotent Header Files" on page 72.) Note that not all compilers support the definitions-separate model for templates.

Because a separate definitions file is a header file, it might be included implicitly in many files. It therefore should not contain any function or variable definitions, unless they are part of a template definition. A separate definitions file can include type definitions, including typedefs.

**Note** – Although source-file extensions for template definition files are commonly used (that is, .c, .C, .cc, .cpp, .cxx, or .c++), template definition files are header files. The compiler includes them automatically if necessary. Template definition files should not be compiled independently.

If you place template declarations in one file and template definitions in another file, you have to be very careful how you construct the definition file, what you name it, and where you put it. You might also need to identify explicitly to the compiler the location of the definitions. Refer to "7.5 Template Definition Searching" on page 98" for information about the template definition search rules.
Creating and Using Templates

Templates make it possible for you to write a single body of code that applies to a wide range of types in a type-safe manner. This chapter introduces template concepts and terminology in the context of function templates, discusses the more complicated (and more powerful) class templates, and describes the composition of templates. Also discussed are template instantiation, default template parameters, and template specialization. The chapter concludes with a discussion of potential problem areas for templates.

6.1 Function Templates

A function template describes a set of related functions that differ only by the types of their arguments or return values.

6.1.1 Function Template Declaration

You must declare a template before you can use it. A declaration, as in the following example, provides enough information to use the template, but not enough information to implement the template.

```cpp
template <class Number> Number twice( Number original );
```

In this example, `Number` is a template parameter; it specifies the range of functions that the template describes. More specifically, `Number` is a template type parameter, and its use within the template definition stands for a type determined at the location where the template is used.

6.1.2 Function Template Definition

If you declare a template, you must also define it. A definition provides enough information to implement the template. The following example defines the template declared in the previous example.
template <class Number> Number twice( Number original )
{ return original + original; }

Because template definitions often appear in header files, a template definition might be repeated in several compilation units. All definitions, however, must be the same. This restriction is called the One-Definition Rule.

6.1.3 Function Template Use

Once declared, templates can be used like any other function. Their use consists of naming the template and providing function arguments. The compiler can infer the template type arguments from the function argument types. For example, you can use the previously declared template as follows.

double twicedouble( double item )
{ return twice( item ); }

If a template argument cannot be inferred from the function argument types, it must be supplied where the function is called. For example:

template<class T> T func(); // no function arguments
int k = func<int>(); // template argument supplied explicitly

6.2 Class Templates

A class template describes a set of related classes or data types that differ only by types, by integral values, by pointers or references to variables with global linkage, or by a combination thereof. Class templates are particularly useful in describing generic, but type-safe, data structures.

6.2.1 Class Template Declaration

A class template declaration provides only the name of the class and its template arguments. Such a declaration is an incomplete class template.

The following example is a template declaration for a class named Array that takes any type as an argument.

template <class Elem> class Array;

This template is for a class named String that takes an unsigned int as an argument.

template <unsigned Size> class String;
6.2.2 Class Template Definition

A class template definition must declare the class data and function members, as in the following examples.

```cpp
template <class Elem> class Array {
    Elem* data;
    int size;
public:
    Array( int sz );
    int GetSize();
    Elem& operator[]( int idx );
};

template <unsigned Size> class String {
    char data[Size];
    static int overflows;
public:
    String( char *initial );
    int length();
};
```

Unlike function templates, class templates can have both type parameters (such as `class Elem`) and expression parameters (such as `unsigned Size`). An expression parameter can be:

- A value that has an integral type or enumeration
- A pointer or a reference to an object
- A pointer or a reference to a function
- A pointer to a class member function

6.2.3 Class Template Member Definitions

The full definition of a class template requires definitions for its function members and static data members. Dynamic (nonstatic) data members are sufficiently defined by the class template declaration.

6.2.3.1 Function Member Definitions

The definition of a template function member consists of the template parameter specification followed by a function definition. The function identifier is qualified by the class template’s class name and the template arguments. The following example shows definitions of two function members of the `Array` class template, which has a template parameter specification of `template <class Elem>`. Each function identifier is qualified by the template class name and the template argument `Array< Elem>`. 
6.2 Class Templates

template <class Elem> Array<Elem>::Array( int sz )
    {size = sz; data = new Elem[size];}

template <class Elem> int Array<Elem>::GetSize()
    { return size; }

This example shows definitions of function members of the String class template.

#include <string.h>
template <unsigned Size> int String<Size>::length( )
    {int len = 0;
     while (len < Size && data[len]!=\0) len++;
     return len;}
template <unsigned Size> String<Size>::String(char *initial)
    {strncpy(data, initial, Size);
     if (length( ) == Size) overflows++;}

6.2.3.2 Static Data Member Definitions

The definition of a template static data member consists of the template parameter specification followed by a variable definition, where the variable identifier is qualified by the class template name and its template actual arguments.

template <unsigned Size> int String<Size>::overflows = 0;

6.2.4 Class Template Use

A template class can be used wherever a type can be used. Specifying a template class consists of providing the values for the template name and arguments. The declaration in the following example creates the variable int_array based upon the Array template. The variable's class declaration and its set of methods are just like those in the Array template except that Elem is replaced with int (see "6.3 Template Instantiation" on page 79).

Array<int> int_array(100);

The declaration in this example creates the short_string variable using the String template.

String<8> short_string("hello");

You can use template class member functions as you would any other member function

int x = int_array.GetSize();

int x = short_string.length();
6.3 Template Instantiation

Template instantiation involves generating a concrete class or function (instance) for a particular combination of template arguments. For example, the compiler generates a class for `Array<int>` and a different class for `Array<double>`. The new classes are defined by substituting the template arguments for the template parameters in the definition of the template class. In the `Array<int>` example, shown in the preceding "Class Templates" section, the compiler substitutes `int` wherever `Elem` appears.

6.3.1 Implicit Template Instantiation

The use of a template function or template class introduces the need for an instance. If that instance does not already exist, the compiler implicitly instantiates the template for that combination of template arguments.

6.3.2 Explicit Template Instantiation

The compiler implicitly instantiates templates only for those combinations of template arguments that are actually used. This approach may be inappropriate for the construction of libraries that provide templates. C++ provides a facility to explicitly instantiate templates, as seen in the following examples.

6.3.2.1 Explicit Instantiation of Template Functions

To instantiate a template function explicitly, follow the template keyword by a declaration (not definition) for the function, with the function identifier followed by the template arguments.

```cpp
template float twice<float>(float original);
```

Template arguments may be omitted when the compiler can infer them.

```cpp
template int twice(int original);
```

6.3.2.2 Explicit Instantiation of Template Classes

To instantiate a template class explicitly, follow the template keyword by a declaration (not definition) for the class, with the class identifier followed by the template arguments.

```cpp
template class Array<char>;
template class String<19>;
```

When you explicitly instantiate a class, all of its members are also instantiated.
6.3.2.3 Explicit Instantiation of Template Class Function Members

To explicitly instantiate a template class function member, follow the `template` keyword by a declaration (not definition) for the function, with the function identifier qualified by the template class, followed by the template arguments.

```
template int Array<char>::GetSize();
```

```
template int String<19>::length();
```

6.3.2.4 Explicit Instantiation of Template Class Static Data Members

To explicitly instantiate a template class static data member, follow the `template` keyword by a declaration (not definition) for the member, with the member identifier qualified by the template class, followed by the template argument.

```
template int String<19>::overflows;
```

6.4 Template Composition

You can use templates in a nested manner. This is particularly useful when defining generic functions over generic data structures, as in the standard C++ library. For example, a template sort function may be declared over a template array class:

```c++
template <class Elem> void sort(Array<Elem>);
```

and defined as:

```c++
template <class Elem> void sort(Array<Elem> store)
{
    int num_elems = store.GetSize();
    for (int i = 0; i < num_elems-1; i++)
        for (int j = i+1; j < num_elems; j++)
            if (store[j-1] > store[j])
                {Elem temp = store[j];
                 store[j] = store[j-1];
                 store[j-1] = temp;}
}
```

The preceding example defines a sort function over the predeclared `Array` class template objects. The next example shows the actual use of the sort function.

```c++
Array<int> int_array(100);    // construct an array of ints
sort(int_array);             // sort it
```
6.5 Default Template Parameters

You can give default values to template parameters for class templates (but not function templates).

```cpp
template <class Elem = int> class Array;
template <unsigned Size = 100> class String;
```

If a template parameter has a default value, all parameters after it must also have default values. A template parameter can have only one default value.

6.6 Template Specialization

There may be performance advantages to treating some combinations of template arguments as a special case, as in the following examples for twice. Alternatively, a template description might fail to work for a set of its possible arguments, as in the following examples for sort. Template specialization allows you to define alternative implementations for a given combination of actual template arguments. The template specialization overrides the default instantiation.

6.6.1 Template Specialization Declaration

You must declare a specialization before any use of that combination of template arguments. The following examples declare specialized implementations of twice and sort.

```cpp
template <> unsigned twice<unsigned>( unsigned original );
template <> sort<char*>(Array<char*> store);
```

You can omit the template arguments if the compiler can unambiguously determine them. For example:

```cpp
template <> unsigned twice(unsigned original);
template <> sort(Array<char*> store);
```

6.6.2 Template Specialization Definition

You must define all template specializations that you declare. The following examples define the functions declared in the preceding section.

```cpp
template <> unsigned twice<unsigned>(unsigned original)
    {return original << 1;}
```
#include <string.h>
template <> void sort<char*>(Array<char*> store)
{
    int num_elems = store.GetSize();
    for (int i = 0; i < num_elems-1; i++)
        for (int j = i+1; j < num_elems; j++)
            if (strcmp(store[j-1], store[j]) > 0)
                {char *temp = store[j];
                 store[j] = store[j-1];
                 store[j-1] = temp;}  

6.6.3 Template Specialization Use and Instantiation

A specialization is used and instantiated just as any other template, except that the definition of a completely specialized template is also an instantiation.

6.6.4 Partial Specialization

In the previous examples, the templates are fully specialized. That is, they define an implementation for specific template arguments. A template can also be partially specialized, meaning that only some of the template parameters are specified, or that one or more parameters are limited to certain categories of type. The resulting partial specialization is itself still a template. For example, the following code sample shows a primary template and a full specialization of that template.

template<class T, class U> class A {...}; //primary template
template<> class A<int, double> {...}; //specialization

The following code shows examples of partial specialization of the primary template.

template<class U> class A<int> {...}; // Example 1
template<class T, class U> class A<T*> {...}; // Example 2
template<class T> class A<T**, char> {...}; // Example 3

- Example 1 provides a special template definition for cases when the first template parameter is type int.
- Example 2 provides a special template definition for cases when the first template parameter is any pointer type.
- Example 3 provides a special template definition for cases when the first template parameter is pointer-to-pointer of any type, and the second template parameter is type char.
6.7 Template Problem Areas

This section describes problems you might encounter when using templates.

6.7.1 Nonlocal Name Resolution and Instantiation

Sometimes a template definition uses names that are not defined by the template arguments or within the template itself. If so, the compiler resolves the name from the scope enclosing the template, which could be the context at the point of definition, or at the point of instantiation. A name can have different meanings in different places, yielding different resolutions.

Name resolution is complex. Consequently, you should not rely on nonlocal names, except those provided in a pervasive global environment. That is, use only nonlocal names that are declared and defined the same way everywhere. In the following example, the template function converter uses the nonlocal names intermediary and temporary. These names have different definitions in use1.cc and use2.cc, and will probably yield different results under different compilers. For templates to work reliably, all nonlocal names (intermediary and temporary in this case) must have the same definition everywhere.

```
use_common.h
// Common template definition
template <class Source, class Target>
Target converter(Source source)
    {temporary = (intermediary)source;
     return (Target)temporary;}
use1.cc
typedef int intermediary;
int temporary;

#include "use_common.h"
use2.cc
typedef double intermediary;
unsigned int temporary;

#include "use_common.h"
```

A common use of nonlocal names is the use of the cin and cout streams within a template. Few programmers really want to pass the stream as a template parameter, so they refer to a global variable. However, cin and cout must have the same definition everywhere.
6.7 Template Problem Areas

6.7.2 Local Types as Template Arguments

The template instantiation system relies on type-name equivalence to determine which templates need to be instantiated or reinstantiated. Thus local types can cause serious problems when used as template arguments. Beware of creating similar problems in your code. For example:

**Example 6-1  Example of Local Type as Template Argument Problem**

```c
array.h
template <class Type> class Array {
    Type* data;
    int size;
public:
    Array(int sz);
    int GetSize();
};

array.cc
template <class Type> Array<Type>::Array(int sz)
    {size = sz; data = new Type[size];}
template <class Type> int Array<Type>::GetSize()
    {return size;}

file1.cc
#include "array.h"
struct Foo {int data;};
Array<Foo> File1Data(10);

file2.cc
#include "array.h"
struct Foo {double data;};
Array<Foo> File2Data(20);
```

The Foo type as registered in file1.cc is not the same as the Foo type registered in file2.cc. Using local types in this way could lead to errors and unexpected results.

6.7.3 Friend Declarations of Template Functions

Templates must be declared before they are used. A friend declaration constitutes a use of the template, not a declaration of the template. A true template declaration must precede the friend declaration. For example, when the compilation system attempts to link the produced object file for the following example, it generates an undefined error for the `operator<<` function, which is not instantiated.
EXAMPLE 6–2  Example of Friend Declaration Problem

array.h
// generates undefined error for the operator<< function
#ifndef ARRAY_H
#define ARRAY_H
#include <iosfwd>

template<class T> class array {
  int size;
public:
  array();
  friend std::ostream&
    operator<< (std::ostream&, const array<T>&); |
};
#endif

array.cc
#include <stdlib.h>
#include <iostream>

template<class T> array<T>::array() {size = 1024;}

template<class T>
std::ostream&
operator<<(std::ostream& out, const array<T>& rhs)
  {return out << '[' << rhs.size << ']';}

main.cc
#include <iostream>
#include "array.h"

int main()
{
  std::cout
    << "creating an array of int... " << std::flush;
  array<int> foo;
  std::cout << "done\n";
  std::cout << foo << std::endl;
  return 0;
}

Note that there is no error message during compilation because the compiler reads the following as the declaration of a normal function that is a friend of the array class.

friend ostream& operator<<(ostream&, const array<T>&);
Because `operator` is really a template function, you need to supply a template declaration for prior to the declaration of template class `array`. However, because `operator` has a parameter of type `array<T>`, you must precede the function declaration with a declaration of `array<T>`. The file `array.h` must look like this:

```cpp
#ifndef ARRAY_H
#define ARRAY_H
#include <iosfwd>

// the next two lines declare operator as a template function
template<class T> class array;
template<class T>
std::ostream& operator<<(std::ostream&, const array<T>&);

template<class T> class array {
    int size;
public:
    array();
    friend std::ostream&
        operator<<(std::ostream&, const array<T>&);
};
#endif
```

### 6.7.4 Using Qualified Names Within Template Definitions

The C++ standard requires types with qualified names that depend upon template arguments to be explicitly noted as type names with the `typename` keyword. This is true even if the compiler can "know" that it should be a type. The comments in the following example show the types with qualified names that require the `typename` keyword.

```cpp
struct simple {
    typedef int a_type;
    static int a_datum;
};
int simple::a_datum = 0; // not a type

template <class T> struct parametric {
    typedef T a_type;
    static T a_datum;
};
template <class T> T parametric<T>::a_datum = 0; // not a type

template <class T> struct example {
    static typename T::a_type variable1; // dependent
    static typename parametric<T>::a_type variable2; // dependent
    static simple::a_type variable3; // not dependent
};
template <class T> typename T::a_type // dependent
```
6.7.5 Nesting Template Names

Because the ">>" character sequence is interpreted as the right-shift operator, you must be careful when you use one template names inside another. Make sure you separate adjacent ">>" characters with at least one blank space.

For example, the following ill-formed statement:

```cpp
def example<T>::variable1 = 0;  // not a type
template <class T> typename parametric<T>::a_type // dependent
def example<T>::variable2 = 0;  // not a type
template <class T> simple::a_type // not dependent
def example<T>::variable3 = 0;  // not a type
```

```cpp
Array<String<10>> short_string_array(100); // >> = right-shift
```

is interpreted as:

```cpp
Array<String<10>> short_string_array(100);
```

The correct syntax is:

```cpp
Array<String<10> > short_string_array(100);
```

6.7.6 Referencing Static Variables and Static Functions

Within a template definition, the compiler does not support referencing an object or function that is declared static at global scope or in a namespace. If multiple instances are generated, the One-Definition Rule (C++ standard section 3.2) is violated, because each instance refers to a different object. The usual failure indication is missing symbols at link time.

If you want a single object to be shared by all template instantiations, then make the object a nonstatic member of a named namespace. If you want a different object for each instantiation of a template class, then make the object a static member of the template class. If you want a different object for each instantiation of a template function, then make the object local to the function.

6.7.7 Building Multiple Programs Using Templates in the Same Directory

If you are building more than one program or library by specifying `-instances=extern`, it's advisable to build them in separate directories. If you want to build in the same directory then you should clean the repository between the different builds. This avoids any unpredictable errors. For more information see “7.4.4 Sharing Template Repositories” on page 97.
Consider the following example with make files a.cc, b.cc, x.h, and x.cc. Note that this example is meaningful only if you specify -instances=extern:

........
Makefile
........
CCC = CC

all: a b

a:

$(CCC) -I. -instances=extern -c a.cc
$(CCC) -instances=extern -o a a.o

b:

$(CCC) -I. -instances=extern -c b.cc
$(CCC) -instances=extern -o b b.o

clean:

/bin/rm -rf SunWS_cache *.o a b

...
x.h
...
template <class T> class X {
public:
    int open();
    int create();
    static int variable;
};
...
x.cc
...
template <class T> int X<T>::create() {
    return variable;
}
template <class T> int X<T>::open() {
    return variable;
}
template <class T> int X<T>::variable = 1;
...
a.cc
...
#include "x.h"
int main()
{
    X<int> templ;
    templ.open();
    templ.create();
}

... b.cc ...
#include "x.h"

int main()
{
    X<int> templ;
    templ.create();
}

If you build both a and b, add a make clean between the two builds. The following commands result in an error:

eexample% make a
eexample% make b

The following commands will not produce any error:

eexample% make a
eexample% make clean
eexample% make b
Template compilation requires the C++ compiler to do more than traditional UNIX compilers have done. The C++ compiler must generate object code for template instances on an as-needed basis. It might share template instances among separate compilations using a template repository. It might accept some template compilation options. It must locate template definitions in separate source files and maintain consistency between template instances and mainline code.

7.1 Verbose Compilation

When given the flag -verbose=template, the C++ compiler notifies you of significant events during template compilation. Conversely, the compiler does not notify you when given the default, -verbose=no%template. The +w option might give other indications of potential problems when template instantiation occurs.

7.2 Repository Administration

The CAdmin(1) command administers the template repository (used only with the option -instances=extern). For example, changes in your program can render some instantiations superfluous, thus wasting storage space. The CAdmin--clean command (formerly ptcLean) clears out all instantiations and associated data. Instantiations are recreated only when needed.

7.2.1 Generated Instances

The compiler treats inline template functions as inline functions for the purposes of template instance generation. The compiler manages them as it does other inline functions, and the descriptions in this chapter do not apply to template inline functions.
7.2.2 Whole-Class Instantiation

The compiler usually instantiates members of template classes independently of other members, so that the compiler instantiates only members that are used within the program. Methods written solely for use through a debugger will therefore not normally be instantiated.

There are two means to ensure that debugging members are available to the debugger.

- First, write a non-template function that uses the template class instance members that are otherwise unused. This function need not be called.
- Second, use the `-template=wholeclass` compiler option, which instructs the compiler to instantiate all non-template non-inline members of a template class if any of those same members are instantiated.

The ISO C++ Standard permits developers to write template classes for which all members may not be legal with a given template argument. As long as the illegal members are not instantiated, the program is still well formed. The ISO C++ Standard Library uses this technique. However, the `-template=wholeclass` option instantiates all members, and hence cannot be used with such template classes when instantiated with the problematic template arguments.

7.2.3 Compile-Time Instantiation

Instantiation is the process by which a C++ compiler creates a usable function or object from a template. The C++ compiler uses compile-time instantiation, which forces instantiations to occur when the reference to the template is being compiled.

The advantages of compile-time instantiation are:

- Debugging is much easier—error messages occur within context, allowing the compiler to give a complete traceback to the point of reference.
- Template instantiations are always up-to-date.
- The overall compilation time, including the link phase, is reduced.

Templates can be instantiated multiple times if source files reside in different directories or if you use libraries with template symbols.

7.2.4 Template Instance Placement and Linkage

By default, instances go into special address sections, and the linker recognizes and discards duplicates. You can instruct the compiler to use one of five instance placement and linkage methods: external, static, global, explicit, and semi-explicit.

- External instances perform best when the following is true:
  - The set of instances in the program is small, but each compilation unit references a large subset of the instances.
There are few instances referenced in more than one or two compilation units.
Static, deprecated. See below.

Global instances, the default, are suitable for all development, and perform best when objects reference a variety of instances.

Explicit instances are suitable for some carefully controlled application compilation environments.

Semi-explicit instances require slightly less controlled compilation environments but produce larger object files and have restricted uses.

This section discusses the five instance placement and linkage methods. Additional information about generating instances can be found in "6.3 Template Instantiation" on page 79.

7.3 External Instances

With the external instances method, all instances are placed within the template repository. The compiler ensures that exactly one consistent template instance exists; instances are neither undefined nor multiply defined. Templates are reinstantiated only when necessary. For non-debug code, the total size of all object files (including any within the template cache) may be smaller with -instances=extern than with -instances=global.

Template instances receive global linkage in the repository. Instances are referenced from the current compilation unit with external linkage.

Note – If you are compiling and linking in separate steps and you specify -instance=extern for the compilation step, you must also specify it for the link step.

The disadvantage of this method is that the cache must be cleared whenever changing programs or making significant program changes. The cache is a bottleneck for parallel compilation, as when using dmake because access to the cache must be restricted to one compilation at a time. Also, you can only build one program within a directory.

It can take longer to determine whether a valid template instance is already in the cache than just to create the instance in the main object file and discard it later if needed.

Specify external linkage with the -instances=extern option.

Because instances are stored within the template repository, you must use the CC command to link C++ objects that use external instances into programs.

If you wish to create a library that contains all template instances that it uses, use the CC command with the -xar option. Do not use the ar command. For example:

example% CC -xar -instances=extern - o libmain.a a.o b.o c.o
7.3 External Instances

See Table 15–3 for more information.

7.3.1 Possible Cache Conflicts

Do not run different compiler versions in the same directory due to possible cache conflicts when you specify -instance=extern. Consider the following when you use the -instances=extern template model:

- Do not create unrelated binaries in the same directory. Any binaries (.o,.a,.so, executable programs) created in the same directory should be related, in that names of all objects, functions, and types common to two or more object files have identical definitions.

- It is safe to run multiple compilations simultaneously in the same directory, such as when using dmake. It is not safe to run any compilations or link steps at the same time as another link step. "Link step" means any operation that creates a library or executable program. Be sure that dependencies in a makefile do not allow anything to run in parallel with a link step.

7.3.2 Static Instances

Note – The -instances=static option is deprecated. There is no longer any reason to use -instances=static, because -instances=global now gives you all the advantages of static without the disadvantages. This option was provided in earlier compilers to overcome problems that no longer exist.

With the static instances method, all instances are placed within the current compilation unit. As a consequence, templates are reinstantiated during each recompilation; instances are not saved to the template repository.

The disadvantage of this method is that it does not follow language semantics and makes substantially larger objects and executables.

Instances receive static linkage. These instances will not be visible or usable outside the current compilation unit. As a result, templates might have identical instantiations in several object files. Because multiple instances produce unnecessarily large programs, static instance linkage is suitable only for small programs, where templates are unlikely to be multiply instantiated.

Compilation is potentially faster with static instances, so this method might also be suitable during Fix-and-Continue debugging. (See Debugging a Program With dbx.)
Note – If your program depends on sharing template instances (such as static data members of template classes or template functions) across compilation units, do not use the static instances method. Your program will not work properly.

Specify static instance linkage with the `-instances=static` compiler option.

### 7.3.3 Global Instances

Unlike early compiler releases, it is not necessary to guard against multiple copies of a global instance.

The advantage of this method is that incorrect source code commonly accepted by other compilers is now also accepted in this mode. In particular, references to static variables from within a template instance are not legal, but commonly accepted.

The disadvantage of this method is that individual object files may be larger, due to copies of template instances in multiple files. If you compile some object files for debug using the `-g` option, and some without, it is hard to predict whether you will get a debug or non-debug version of a template instance linked into the program.

Template instances receive global linkage. These instances are visible and usable outside the current compilation unit.

Specify global instances with the `-instances=global` option (this is the default).

### 7.3.4 Explicit Instances

In the explicit instances method, instances are generated only for templates that are explicitly instantiated. Implicit instantiations are not satisfied. Instances are placed within the current compilation unit.

The advantage of this method is that you have the least amount of template compilation and smallest object sizes.

The disadvantage is that you must perform all instantiation manually.

Template instances receive global linkage. These instances are visible and usable outside the current compilation unit. The linker recognizes and discards duplicates.

Specify explicit instances with the `-instances=explicit` option.
7.3.5 Semi-Explicit Instances

When you use the semi-explicit instances method, instances are generated only for templates that are explicitly instantiated or implicitly instantiated within the body of a template. Instances required by explicitly-created instances are generated automatically. Implicit instantiations in the mainline code are not satisfied. Instances are placed within the current compilation unit. As a consequence, templates are reinstantiated during each recompilation; instances receive global linkage and they are not saved to the template repository.

Specify semi-explicit instances with the -instances=semiexplicit option.

7.4 The Template Repository

The template repository stores template instances between separate compilations so that template instances are compiled only when it is necessary. The template repository contains all nonsource files needed for template instantiation when using the external instances method. The repository is not used for other kinds of instances.

7.4.1 Repository Structure

The template repository is contained, by default, within a cache directory called SunWS_cache. The cache directory is contained within the directory in which the object files are placed. You can change the name of the cache directory by setting the SUNWS_CACHE_NAME environment variable. Note that the value of the SUNWS_CACHE_NAME variable must be a directory name and not a path name. This is because the compiler automatically places the template cache directory under the object file directory so the compiler already has a path.

7.4.2 Writing to the Template Repository

When the compiler must store template instances, it stores them within the template repository corresponding to the output file. For example, the following command line writes the object file to ./sub/a.o and writes template instances into the repository contained within ./sub/SunWS_cache. If the cache directory does not exist, and the compiler needs to instantiate a template, the compiler will create the directory.

example% CC -o sub/a.o a.cc
7.4.3 **Reading From Multiple Template Repositories**

The compiler reads from the template repositories corresponding to the object files that it reads. That is, the following command line reads from ./sub1/SunWS_cache and ./sub2/SunWS_cache, and, if necessary, writes to ./SunWS_cache.

```bash
example% CC sub1/a.o sub2/b.o
```

7.4.4 **Sharing Template Repositories**

Templates that are within a repository must not violate the one-definition rule of the ISO C++ standard. That is, a template must have the same source in all uses of the template. Violating this rule produces undefined behavior.

The simplest, though most conservative, way to ensure that the rule is not violated is to build only one program or library within any one directory. Two unrelated programs might use the same type name or external name to mean different things. If the programs share a template repository, template definitions could conflict, thus yielding unpredictable results.

7.4.5 **Template Instance Automatic Consistency With -instances=extern**

The template repository manager ensures that the states of the instances in the repository are consistent and up-to-date with your source files when you specify `-instances=extern`.

For example, if your source files are compiled with the `–g` option (debugging on), the files you need from the database are also compiled with `–g`.

In addition, the template repository tracks changes in your compilation. For example, if you have the `–DDEBUG` flag set to define the name `DEBUG`, the database tracks this. If you omit this flag on a subsequent compile, the compiler reinstantiates those templates on which this dependency is set.

**Note** - If you remove the source code of a template or stop using a template, instances of the template remain in the cache. If you change the signature of a function template, instances using the old signature remain in the cache. If you run into strange behavior at compile or link time due to these issues, clear the template cache and rebuild the program.
7.5 Template Definition Searching

When you use the definitions-separate template organization, template definitions are not available in the current compilation unit, and the compiler must search for the definition. This section describes how the compiler locates the definition.

Definition searching is somewhat complex and prone to error. Therefore, you should use the definitions-included template file organization if possible. Doing so helps you avoid definition searching altogether. See "5.2.1 Template Definitions Included" on page 73.

Note – If you use the -template=no%extdef option, the compiler will not search for separate source files.

7.5.1 Source File Location Conventions

Without the specific directions provided with an options file, the compiler uses a Cfront-style method to locate template definition files. This method requires that the template definition file contain the same basename as the template declaration file. This method also requires that the template definition file be on the current include path. For example, if the template function foo() is located in foo.h, the matching template definition file should be named foo.cc or some other recognizable source-file extension (.C, .c, .cc, .cpp, .cxx, or .c++). The template definition file must be located in one of the normal include directories or in the same directory as its matching header file.

7.5.2 Definitions Search Path

As an alternative to the normal search path set with –I, you can specify a search directory for template definition files with the option –ptidirectory. Multiple -pti flags define multiple search directories—that is, a search path. If you use -ptidirectory, the compiler looks for template definition files on this path and ignores the –I flag. Since the –ptidirectory flag complicates the search rules for source files, use the –I option instead of the –ptidirectory option.

7.5.3 Troubleshooting a Problematic Search

Sometimes the compiler generates confusing warnings or error messages because it is looking for file that you don’t intend to compile. Usually, the problem is that a file, for example foo.h, contains template declarations and another file, such as foo.cc, gets implicitly included.
If a header file, `foo.h`, has template declarations, the compiler searches for a file called `foo` with a C++ file extension (`.C`, `.c`, `.cc`, `.cpp`, `.cxx`, or `.c++`) by default. If the compiler finds such a file, it includes the file automatically. See “7.5 Template Definition Searching” on page 98 for more information on such searches.

If you have a file `foo.cc` that you don’t intend to be treated this way, you have two options:

- Change the name of the `.h` or the `.cc` file to eliminate the name match.
- Disable the automatic search for template definition files by specifying the `-template=no%extdef` option. You must then include all template definitions explicitly in your code and will not be able to use the “definitions separate” model.
This chapter discusses the C++ compiler’s implementation of exception handling. Additional information can be found in “11.2 Using Exceptions in a Multithreaded Program” on page 118. For more information on exception handling, see *The C++ Programming Language*, Third Edition, by Bjarne Stroustrup (Addison-Wesley, 1997).

### 8.1 Synchronous and Asynchronous Exceptions

Exception handling is designed to support only synchronous exceptions, such as array range checks. The term *synchronous exception* means that exceptions can be originated only from throw expressions.

The C++ standard supports synchronous exception handling with a termination model. *Termination* means that once an exception is thrown, control never returns to the throw point.

Exception handling is not designed to directly handle asynchronous exceptions such as keyboard interrupts. However, you can make exception handling work in the presence of asynchronous events if you are careful. For instance, to make exception handling work with signals, you can write a signal handler that sets a global variable, and create another routine that polls the value of that variable at regular intervals and throws an exception when the value changes. You cannot throw an exception from a signal handler.

### 8.2 Specifying Runtime Errors

There are five runtime error messages associated with exceptions:

- No handler for the exception
- Unexpected exception thrown
- An exception can only be re-thrown in a handler
- During stack unwinding, a destructor must handle its own exception
When errors are detected at runtime, the error message displays the type of the current exception and one of the five error messages. By default, the predefined function `terminate()` is called, which then calls `abort()`.

The compiler uses the information provided in the exception specification to optimize code production. For example, table entries for functions that do not throw exceptions are suppressed, and runtime checking for exception specifications of functions is eliminated wherever possible.

### 8.3 Disabling Exceptions

If you know that exceptions are not used in a program, you can use the compiler option `-features=no%except` to suppress generation of code that supports exception handling. The use of the option results in slightly smaller code size and faster code execution. However, when files compiled with exceptions disabled are linked to files using exceptions, some local objects in the files compiled with exceptions disabled are not destroyed when exceptions occur. By default, the compiler generates code to support exception handling. Unless the time and space overhead is important, it is usually better to leave exceptions enabled.

**Note** – Because the C++ standard library, `dynamic_cast`, and the default operator `new` require exceptions, you should not turn off exceptions when you compile in standard mode (the default mode).

### 8.4 Using Runtime Functions and Predefined Exceptions

The standard header `<exception>` provides the classes and exception-related functions specified in the C++ standard. You can access this header only when compiling in standard mode (compiler default mode, or with option `-compat=5`). The following excerpt shows the `<exception>` header file declarations.

```cpp
// standard header <exception>
namespace std {
  class exception {
    exception() throw();
    exception(const exception&) throw();
    exception& operator=(const exception&) throw();
    virtual ~exception() throw();
    virtual const char* what() const throw();
  };
  class bad_exception: public exception {...};
```
// Unexpected exception handling
typedef void (*unexpected_handler)();
unexpected_handler
    set_unexpected(unexpected_handler) throw();
void unexpected();

// Termination handling
typedef void (*terminate_handler)();
terminate_handler set_terminate(terminate_handler) throw();
void terminate();
bool uncaught_exception() throw();

The standard class exception is the base class for all exceptions thrown by selected language constructs or by the C++ standard library. An object of type exception can be constructed, copied, and destroyed without generating an exception. The virtual member function what() returns a character string that describes the exception.

For compatibility with exceptions as used in C++ release 4.2, the header <exception.h> is also provided for use in standard mode. This header allows for a transition to standard C++ code and contains declarations that are not part of standard C++. Update your code to follow the C++ standard (using <exception> instead of <exception.h>) as development schedules permit.

// header <exception.h>, used for transition
#include <exception>
#include <new>
using std::exception;
using std::bad_exception;
using std::set_unexpected;
using std::unexpected;
using std::set_terminate;
using std::terminate;
typedef std::exception xmsg;
typedef std::bad_exception xunexpected;
typedef std::bad_alloc xalloc;

In compatibility mode (~compat[=4]), header <exception.h> is not available, and header <exception.h> refers to the same header provided with C++ release 4.2. It is not reproduced here.
8.5 Mixing Exceptions With Signals and Setjmp/Longjmp

You can use the setjmp/longjmp functions in a program where exceptions can occur, as long as they do not interact.

All the rules for using exceptions and setjmp/longjmp separately apply. In addition, a longjmp from point A to point B is valid only if an exception thrown at A and caught at B would have the same effect. In particular, you must not longjmp into or out of a try-block or catch-block (directly or indirectly), or longjmp past the initialization or non-trivial destruction of auto variables or temporary variables.

You cannot throw an exception from a signal handler.

8.6 Building Shared Libraries That Have Exceptions

Never use -Bsymbolic with programs containing C++ code, use linker map files instead or linker scoping options (See “4.1 Linker Scoping” on page 63). With -Bsymbolic, references in different modules can bind to different copies of what is supposed to be one global object.

The exception mechanism relies on comparing addresses. If you have two copies of something, their addresses won’t compare equal, and the exception mechanism can fail because the exception mechanism relies on comparing what are supposed to be unique addresses.
This chapter discusses the newer cast operators in the C++ standard: const_cast, reinterpret_cast, static_cast, and dynamic_cast. A cast converts an object or value from one type to another.

These cast operations provide finer control than previous cast operations. The dynamic_cast<> operator provides a way to check the actual type of a pointer to a polymorphic class. You can search with a text editor for all new-style casts (search for _cast), whereas finding old-style casts required syntactic analysis.

Otherwise, the new casts all perform a subset of the casts allowed by the classic cast notation. For example, const_cast<int*>(v) could be written (int*)v. The new casts simply categorize the variety of operations available to express your intent more clearly and allow the compiler to provide better checking.

The cast operators are always enabled. They cannot be disabled.

9.1 const_cast

The expression const_cast<T>(v) can be used to change the const or volatile qualifiers of pointers or references. (Among new-style casts, only const_cast<> can remove const qualifiers.) T must be a pointer, reference, or pointer-to-member type.

class A
{
    public:
        virtual void f();
        int i;
    }
extern const volatile int* cvip;
extern int* ip;
void use_of_const_cast()
9.2 reinterpret_cast

The expression reinterpret_cast<T>(v) changes the interpretation of the value of the expression v. It can be used to convert between pointer and integer types, between unrelated pointer types, between pointer-to-member types, and between pointer-to-function types.

Usage of the reinterpret_cast operator can have undefined or implementation-dependent results. The following points describe the only ensured behavior:

- A pointer to a data object or to a function (but not a pointer to member) can be converted to any integer type large enough to contain it. (Type long is always large enough to contain a pointer value on the architectures supported by the C++ compiler.) When converted back to the original type, the pointer value will compare equal to the original pointer.
- A pointer to a (nonmember) function can be converted to a pointer to a different (nonmember) function type. If converted back to the original type, the pointer value will compare equal to the original pointer.
- A pointer to an object can be converted to a pointer to a different object type, provided that the new type has alignment requirements no stricter than the original type. When converted back to the original type, the pointer value will compare equal to the original pointer.
- An lvalue of type T1 can be converted to a type "reference to T2" if an expression of type "pointer to T1" can be converted to type "pointer to T2" with a reinterpret cast.
- An rvalue of type "pointer to member of X of type T1" can be explicitly converted to an rvalue of type "pointer to member of Y of type T2" if T1 and T2 are both function types or both object types.
- In all allowed cases, a null pointer of one type remains a null pointer when converted to a null pointer of a different type.
- The reinterpret_cast operator cannot be used to cast away const; use const_cast for that purpose.
- The reinterpret_cast operator should not be used to convert between pointers to different classes that are in the same class hierarchy; use a static or dynamic cast for that purpose. (reinterpret_cast does not perform the adjustments that might be needed.) This is illustrated in the following example:

```cpp
class A { int a; public: A();
class B: public A { int b, c;};
void use_of_reinterpret_cast()
```
### 9.4 Dynamic Casts

A pointer (or reference) to a class can actually point (refer) to any class derived from that class. Occasionally, it may be desirable to obtain a pointer to the fully derived class, or to some other subobject of the complete object. The dynamic cast provides this facility.

---

**Note** - When compiling in compatibility mode (`-compat[=4]`), you must compile with `-ffeatures=rtti` if your program uses dynamic casts.

The dynamic type cast converts a pointer (or reference) to one class $T_1$ into a pointer (reference) to another class $T_2$. $T_1$ and $T_2$ must be part of the same hierarchy, the classes must
be accessible (via public derivation), and the conversion must not be ambiguous. In addition, unless the conversion is from a derived class to one of its base classes, the smallest part of the hierarchy enclosing both $T_1$ and $T_2$ must be polymorphic (have at least one virtual function).

In the expression `dynamic_cast<T>(v)`, $v$ is the expression to be cast, and $T$ is the type to which it should be cast. $T$ must be a pointer or reference to a complete class type (one for which a definition is visible), or a pointer to `cv void`, where `cv` is an empty string, `const`, `volatile`, or `const volatile`.

### 9.4.1 Casting Up the Hierarchy

When casting up the hierarchy, if $T$ points (or refers) to a base class of the type pointed (referred) to by $v$, the conversion is equivalent to `static_cast<T>(v)`.

### 9.4.2 Casting to `void*`

If $T$ is `void*`, the result is a pointer to the complete object. That is, $v$ might point to one of the base classes of some complete object. In that case, the result of `dynamic_cast<void*>(v)` is the same as if you converted $v$ down the hierarchy to the type of the complete object (whatever that is) and then to `void*`.

When casting to `void*`, the hierarchy must be polymorphic (have virtual functions).

### 9.4.3 Casting Down or Across the Hierarchy

When casting down or across the hierarchy, the hierarchy must be polymorphic (have virtual functions). The result is checked at runtime.

The conversion from $v$ to $T$ is not always possible when casting down or across a hierarchy. For example, the attempted conversion might be ambiguous, $T$ might be inaccessible, or $v$ might not point (or refer) to an object of the necessary type. If the runtime check fails and $T$ is a pointer type, the value of the cast expression is a null pointer of type $T$. If $T$ is a reference type, nothing is returned (there are no null references in C++), and the standard exception `std::bad_cast` is thrown.

For example, this example of public derivation succeeds:

```cpp
#include <assert.h>
#include <stddef.h> // for NULL

class A {public: virtual void f();};
class B {public: virtual void g();};
class AB: public virtual A, public B {};
```
void simple_dynamic_casts()
{
    AB ab;
    B* bp = &ab; // no casts needed
    A* ap = &ab;
    AB& abr = dynamic_cast<AB&>(*bp); // succeeds
    ap = dynamic_cast<A*>(bp); assert(ap != NULL);
    bp = dynamic_cast<B*>(ap); assert(bp != NULL);
    ap = dynamic_cast<A*>(&abr); assert(ap != NULL);
    bp = dynamic_cast<B*>(&abr); assert(bp != NULL);
}

whereas this example fails because base class B is inaccessible.

#include <assert.h>
#include <stddef.h> // for NULL
#include <typeinfo>

class A {public: virtual void f() {}};
class B {public: virtual void g() {}};
class AB: public virtual A, private B {};

void attempted_casts()
{
    AB ab;
    B* bp = (B*)&ab; // C-style cast needed to break protection
    A* ap = dynamic_cast<A*>(bp); // fails, B is inaccessible
    assert(ap == NULL);
    try {
        AB& abr = dynamic_cast<AB&>(*bp); // fails, B is inaccessible
    }
    catch(const std::bad_cast&) {
        return; // failed reference cast caught here
    }
    assert(0); // should not get here
}

In the presence of virtual inheritance and multiple inheritance of a single base class, the actual
dynamic cast must be able to identify a unique match. If the match is not unique, the cast fails.
For example, given the additional class definitions:

class AB_B: public AB, public B {};
class AB_B__AB: public AB_B, public AB {};

Example:

void complex_dynamic_casts()
{

9.4 Dynamic Casts

```cpp
AB_B__AB ab_b__ab;
A*ap = &ab_b__ab;
    // okay: finds unique A statically
AB*abp = dynamic_cast<AB*>(ap);
    // fails: ambiguous
assert(abp == NULL);
    // STATIC ERROR: AB B* ab_bp = (AB_B*)ap;
    // not a dynamic cast
AB_B*ab_bp = dynamic_cast<AB_B*>(ap);
    // dynamic one is okay
assert(ab_bp!= NULL);
}
```

The null-pointer error return of `dynamic_cast` is useful as a condition between two bodies of code—one to handle the cast if the type guess is correct, and one if it is not.

```cpp
void using_dynamic_cast(A* ap)
{
    if (AB *abp = dynamic_cast<AB*>(ap))
    {
        // abp is non-null,
        // so ap was a pointer to an AB object
        // go ahead and use abp
        process_AB(abp);
    }
    else
    {
        // abp is null,
        // so ap was NOT a pointer to an AB object
        // do not use abp
        process_not_AB(ap);
    }
}
```

In compatibility mode (`-compat[=4]`), if runtime type information has not been enabled with the `-features=rtti` compiler option, the compiler converts `dynamic_cast` to `static_cast` and issues a warning.

If exceptions have been disabled, the compiler converts `dynamic_cast<T&>` to `static_cast<T&>` and issues a warning. (A `dynamic_cast` to a reference type requires an exception to be thrown if the conversion is found at run time to be invalid.) For information about exceptions, see "7.5.3 Troubleshooting a Problematic Search" on page 98.

Dynamic cast is necessarily slower than an appropriate design pattern, such as conversion by virtual functions. See *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma (Addison-Wesley, 1994).
You can improve the performance of C++ functions by writing those functions in a manner that helps the compiler do a better job of optimizing them. Many books have been written on software performance in general and C++ in particular, and this chapter does not repeat such valuable information, but discusses only those performance techniques that strongly affect the C++ compiler.

### 10.1 Avoiding Temporary Objects

C++ functions often produce implicit temporary objects, each of which must be created and destroyed. For non-trivial classes, the creation and destruction of temporary objects can be expensive in terms of processing time and memory usage. The C++ compiler does eliminate some temporary objects, but it cannot eliminate all of them.

Write functions to minimize the number of temporary objects as long as your programs remain comprehensible. Techniques include using explicit variables rather than implicit temporary objects and using reference parameters rather than value parameters. Another technique is to implement and use operations such as `+=` rather than implementing and using only `+` and `=`. For example, the first line below introduces a temporary object for the result of `a + b`, while the second line does not.

```cpp
T x = a + b;
T x(a); x += b;
```
10.2 Using Inline Functions

Calls to small and quick functions can be smaller and quicker when expanded inline than when called normally. Conversely, calls to large or slow functions can be larger and slower when expanded inline than when branched to. Furthermore, all calls to an inline function must be recompiled whenever the function definition changes. Consequently, the decision to use inline functions requires considerable care.

Do not use inline functions when you anticipate changes to the function definition and recompiling all callers is expensive. Otherwise, use inline functions when the code to expand the function inline is smaller than the code to call the function or the application performs significantly faster with the function inline.

The compiler cannot inline all function calls, so making the most effective use of function inlining may require some source changes. Use the +w option to learn when function inlining does not occur. In the following situations, the compiler will not inline the function:

- The function contains difficult control constructs, such as loops, switch statements, and try/catch statements. Many times these functions execute the difficult control constructs infrequently. To inline such a function, split the function into two parts, an inner part that contains the difficult control constructs and an outer part that decides whether or not to call the inner part. This technique of separating the infrequent part from the frequent part of a function can improve performance even when the compiler can inline the full function.

- The inline function body is large or complicated. Apparently simple function bodies may be complicated because of calls to other inline functions within the body, or because of implicit constructor and destructor calls (as often occurs in constructors and destructors for derived classes). For such functions, inline expansion rarely provides significant performance improvement, and the function is best left unlined.

- The arguments to an inline function call are large or complicated. The compiler is particularly sensitive when the object for an inline member function call is itself the result of an inline function call. To inline functions with complicated arguments, simply compute the function arguments into local variables and then pass the variables to the function.

10.3 Using Default Operators

If a class definition does not declare a parameterless constructor, a copy constructor, a copy assignment operator, or a destructor, the compiler will implicitly declare them. These are called default operators. A C-like struct has these default operators. When the compiler builds a default operator, it knows a great deal about the work that needs to be done and can produce very good code. This code is often much faster than user-written code because the compiler can take advantage of assembly-level facilities while the programmer usually cannot. So, when the default operators do what is needed, the program should not declare user-defined versions of these operators.
Default operators are inline functions, so do not use default operators when inline functions are inappropriate (see the previous section). Otherwise, default operators are appropriate when:

- The user-written parameterless constructor would only call parameterless constructors for its base objects and member variables. Primitive types effectively have “do nothing” parameterless constructors.
- The user-written copy constructor would simply copy all base objects and member variables.
- The user-written copy assignment operator would simply copy all base objects and member variables.
- The user-written destructor would be empty.

Some C++ programming texts suggest that class programmers always define all operators so that any reader of the code will know that the class programmer did not forget to consider the semantics of the default operators. Obviously, this advice interferes with the optimization discussed above. The resolution of the conflict is to place a comment in the code stating that the class is using the default operator.

### 10.4 Using Value Classes

C++ classes, including structures and unions, are passed and returned by value. For Plain-Old-Data (POD) classes, the C++ compiler is required to pass the struct as would the C compiler. Objects of these classes are passed *directly*. For objects of classes with user-defined copy constructors, the compiler is effectively required to construct a copy of the object, pass a pointer to the copy, and destroy the copy after the return. Objects of these classes are passed *indirectly*. For classes that fall between these two requirements, the compiler can choose. However, this choice affects binary compatibility, so the compiler must choose consistently for every class.

For most compilers, passing objects directly can result in faster execution. This execution improvement is particularly noticeable with small value classes, such as complex numbers or probability values. You can sometimes improve program efficiency by designing classes that are more likely to be passed directly than indirectly.

In compatibility mode (\(-\text{compat } [\neq 4]\)), a class is passed indirectly if it has any one of the following:

- A user-defined constructor
- A virtual function
- A virtual base class
- A base that is passed indirectly
- A non-static data member that is passed indirectly

Otherwise, the class is passed directly.
In standard mode (the default mode), a class is passed indirectly if it has any one of the following:

- A user-defined copy constructor
- A user-defined destructor
- A base that is passed indirectly
- A non-static data member that is passed indirectly

Otherwise, the class is passed directly.

### 10.4.1 Choosing to Pass Classes Directly

To maximize the chance that a class will be passed directly:

- Use default constructors, especially the default copy constructor, where possible.
- Use the default destructor where possible. The default destructor is not virtual, therefore a class with a default destructor should generally not be a base class.
- Avoid virtual functions and virtual bases.

### 10.4.2 Passing Classes Directly on Various Processors

Classes (and unions) that are passed directly by the C++ compiler are passed exactly as the C compiler would pass a struct (or union). However, C++ structs and unions are passed differently on different architectures.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARC V7/V8</td>
<td>Structs and unions are passed and returned by allocating storage within the caller and passing a pointer to that storage. (That is, all structs and unions are passed by reference.)</td>
</tr>
<tr>
<td>SPARC V9</td>
<td>Structs with a size no greater than 16 bytes (32 bytes) are passed (returned) in registers. Unions and all other structs are passed and returned by allocating storage within the caller and passing a pointer to that storage. (That is, small structs are passed in registers; unions and large structs are passed by reference.) As a consequence, small value classes are passed as efficiently as primitive types.</td>
</tr>
<tr>
<td>x86 platforms</td>
<td>Structs and unions are passed by allocating space on the stack and copying the argument onto the stack. Structs and unions are returned by allocating a temporary object in the caller's frame and passing the address of the temporary object as an implicit first parameter.</td>
</tr>
</tbody>
</table>
10.5 Cache Member Variables

Accessing member variables is a common operation in C++ member functions.

The compiler must often load member variables from memory through the this pointer. Because values are being loaded through a pointer, the compiler sometimes cannot determine when a second load must be performed or whether the value loaded before is still valid. In these cases, the compiler must choose the safe, but slow, approach and reload the member variable each time it is accessed.

You can avoid unnecessary memory reloads by explicitly caching the values of member variables in local variables, as follows:

- Declare a local variable and initialize it with the value of the member variable.
- Use the local variable in place of the member variable throughout the function.
- If the local variable changes, assign the final value of the local variable to the member variable. However, this optimization may yield undesired results if the member function calls another member function on that object.

This optimization is most productive when the values can reside in registers, as is the case with primitive types. The optimization may also be productive for memory-based values because the reduced aliasing gives the compiler more opportunity to optimize.

This optimization may be counter productive if the member variable is often passed by reference, either explicitly or implicitly.

On occasion, the desired semantics of a class requires explicit caching of member variables, for instance when there is a potential alias between the current object and one of the member function’s arguments. For example:

```cpp
complex& operator*= (complex& left, complex& right)
{
    left.real = left.real * right.real + left.imag * right.imag;
    left.imag = left.real * right.imag + left.image * right.real;
}
```

will yield unintended results when called with:

```cpp
x*=x;
```
This chapter explains how to build multithreaded programs. It also discusses the use of exceptions, explains how to share C++ Standard Library objects across threads, and describes how to use classic (old) iostreams in a multithreading environment.


See also the OpenMP API User's Guide for information on using OpenMP shared memory parallelization directives to create multithreaded programs.

11.1 Building Multithreaded Programs

All libraries shipped with the C++ compiler are multithreading safe. If you want to build a multithreaded application, or if you want to link your application to a multithreaded library, you must compile and link your program with the -mt option. This option passes \texttt{-D\_REENTRANT} to the preprocessor and passes -lthread in the correct order to \texttt{ld}. For compatibility mode (-compat[=4]), the -mt option ensures that \texttt{libthread} is linked before \texttt{libC}. For standard mode (the default mode), the -mt option ensures that \texttt{libthread} is linked before \texttt{libC\_run}. Use of -mt is recommended a simpler and less error-prone alternative to specifying the macro and library.

11.1.1 Indicating Multithreaded Compilation

You can check whether an application is linked to \texttt{libthread} or not by using the \texttt{l}dd command:

```
example% CC -mt myprog.cc
example% ldd a.out
libm.so.1 => /usr/lib/libm.so.1
libCrun.so.1 => /usr/lib/libCrun.so.1
```
11.1.2 Using C++ Support Libraries With Threads and Signals

The C++ support libraries, libcrun, libiostream, libCstd, and libC are multithread safe but are not async safe. This means that in a multithreaded application, functions available in the support libraries should not be used in signal handlers. Doing so can result in a deadlock situation.

It is not safe to use the following in a signal handler in a multithreaded application:

- Iostreams
- new and delete expressions
- Exceptions

11.2 Using Exceptions in a Multithreaded Program

The current exception-handling implementation is safe for multithreading; exceptions in one thread do not interfere with exceptions in other threads. However, you cannot use exceptions to communicate across threads; an exception thrown from one thread cannot be caught in another.

Each thread can set its own terminate() or unexpected() function. Calling set_terminate() or set_unexpected() in one thread affects only the exceptions in that thread. The default function for terminate() is abort() for any thread (see "8.2 Specifying Runtime Errors" on page 101).

11.2.1 Thread Cancellation

Thread cancellation through a call to pthread_cancel(3T) results in the destruction of automatic (local nonstatic) objects on the stack except when you specify -noex or -features=noexcept.

pthread_cancel(3T) uses the same mechanism as exceptions. When a thread is cancelled, the execution of local destructors is interleaved with the execution of cleanup routines that the user has registered with pthread_cleanup_push(). The local objects for functions called after a particular cleanup routine is registered are destroyed before that routine is executed.
11.3 Sharing C++ Standard Library Objects Between Threads

The C++ Standard Library (libCstd - library=Cstd) is MT-Safe, with the exception of some locales, and it ensures that the internals of the library work properly in a multi-threaded environment. You still need to lock around any library objects that you yourself share between threads. See the man pages for setlocale(3C) and attributes(5).

For example, if you instantiate a string, then create a new thread and pass that string to the thread by reference, then you must lock around write access to that string, since you are explicitly sharing the one string object between threads. (The facilities provided by the library to accomplish this task are described below.)

On the other hand, if you pass the string to the new thread by value, you do not need to worry about locking, even though the strings in the two different threads may be sharing a representation through Rogue Wave’s “copy on write” technology. The library handles that locking automatically. You are only required to lock when making an object available to multiple threads explicitly, either by passing references between threads or by using global or static objects.

The following describes the locking (synchronization) mechanism used internally in the C++ Standard Library to ensure correct behavior in the presence of multiple threads.

Two synchronization classes provide mechanisms for achieving multithreaded safety; _RWSTDMutex and _RWSTDGuard.

The _RWSTDMutex class provides a platform-independent locking mechanism through the following member functions:
- void acquire() — Acquires a lock on self, or blocks until such a lock can be obtained.
- void release() — Releases a lock on self.

```cpp
class _RWSTDMutex
{
 public:
   _RWSTDMutex ();
   ~_RWSTDMutex ();
   void acquire ();
   void release ();
};
```

The _RWSTDGuard class is a convenience wrapper class that encapsulates an object of _RWSTDMutex class. An _RWSTDGuard object attempts to acquire the encapsulated mutex in its constructor (throwing an exception of type std::thread_error, derived from std::exception on error), and releases the mutex in its destructor (the destructor never throws an exception).
class _RWSTDGuard
{
public:
    _RWSTDGuard (_RWSTDMutex&);
    ~_RWSTDGuard ();
};

Additionally, you can use the macro _RWSTD_MT_GUARD(mutex) (formerly _STDGUARD) to conditionally create an object of the _RWSTDGuard class in multithread builds. The object guards the remainder of the code block in which it is defined from being executed by multiple threads simultaneously. In single-threaded builds the macro expands into an empty expression.

The following example illustrates the use of these mechanisms.

#include <rw/stdmutex.h>

// An integer shared among multiple threads.
// int I;

// A mutex used to synchronize updates to I.
// _RWSTDMutex I_mutex;

// Increment I by one. Uses an _RWSTDMutex directly.
//
void increment_I ()
{
    I_mutex.acquire(); // Lock the mutex.
    I++;
    I_mutex.release(); // Unlock the mutex.
}

// Decrement I by one. Uses an _RWSTDGuard.
//
void decrement_I ()
{
    _RWSTDGuard guard(I_mutex); // Acquire the lock on I_mutex.
    --I;
    // The lock on I is released when destructor is called on guard.
}
11.4 Using Classic iostreams in a Multithreading Environment

This section describes how to use the iostream classes of the libc and libiostream libraries for input-output (I/O) in a multithreaded environment. It also provides examples of how to extend functionality of the library by deriving from the iostream classes. This section is not a guide for writing multithreaded code in C++, however.

The discussion here applies only to the old iostreams (libc and libiostream) and does not apply to libCstd, the new iostream that is part of the C++ Standard Library.

The iostream library allows its interfaces to be used by applications in a multithreaded environment by programs that utilize the multithreading capabilities when running supported versions of the Solaris operating system. Applications that utilize the single-threaded capabilities of previous versions of the library are not affected.

A library is defined to be MT-safe if it works correctly in an environment with threads. Generally, this “correctness” means that all of its public functions are reentrant. The iostream library provides protection against multiple threads that attempt to modify the state of objects (that is, instances of a C++ class) shared by more than one thread. However, the scope of MT-safety for an iostream object is confined to the period in which the object’s public member function is executing.

Note – An application is not automatically guaranteed to be MT-safe because it uses MT-safe objects from the libc library. An application is defined to be MT-safe only when it executes as expected in a multithreaded environment.

11.4.1 Organization of the MT-Safe iostream Library

The organization of the MT-safe iostream library is slightly different from other versions of the iostream library. The exported interface of the library refers to the public and protected member functions of the iostream classes and the set of base classes available, and is consistent with other versions; however, the class hierarchy is different. See “11.4.2 Interface Changes to the iostream Library” on page 127 for details.

The original core classes have been renamed with the prefix unsafe_. Table 11–1 lists the classes that are the core of the iostream package.
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream_MT</td>
<td>The base class for MT-safe classes.</td>
</tr>
<tr>
<td>streambuf</td>
<td>The base class for buffers.</td>
</tr>
<tr>
<td>unsafe_ios</td>
<td>A class that contains state variables that are common to the various stream classes; for example, error and formatting state.</td>
</tr>
<tr>
<td>unsafe_istream</td>
<td>A class that supports formatted and unformatted conversion from sequences of characters retrieved from the streambufs.</td>
</tr>
<tr>
<td>unsafe_ostream</td>
<td>A class that supports formatted and unformatted conversion to sequences of characters stored into the streambufs.</td>
</tr>
<tr>
<td>unsafe_iostream</td>
<td>A class that combines unsafe_istream and unsafe_ostream classes for bidirectional operations.</td>
</tr>
</tbody>
</table>

Each MT-safe class is derived from the base class stream_MT. Each MT-safe class, except streambuf, is also derived from the existing unsafe_base class. Here are some examples:

```cpp
class streambuf: public stream_MT {...};
class ios: virtual public unsafe_ios, public stream_MT {...};
class istream: virtual public ios, public unsafe_istream {...};
```

The class stream_MT provides the mutual exclusion (mutex) locks required to make each iostream class MT-safe; it also provides a facility that dynamically enables and disables the locks so that the MT-safe property can be dynamically changed. The basic functionality for I/O conversion and buffer management are organized into the unsafe_classes; the MT-safe additions to the library are confined to the derived classes. The MT-safe version of each class contains the same protected and public member functions as the unsafe_base class. Each member function in the MT-safe version class acts as a wrapper that locks the object, calls the same function in the unsafe_base class, and unlocks the object.

**Note** – The class streambuf is not derived from an unsafe class. The public and protected member functions of class streambuf are reentrant by locking. Unlocked versions, suffixed with _unlocked, are also provided.

### 11.4.1.1 Public Conversion Routines

A set of reentrant public functions that are MT-safe have been added to the iostream interface. A user-specified buffer is an additional argument to each function. These functions are described as follows.
### MT-Safe Reentrant Public Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *oct_r (char *buf, int buflen, long num, int width)</td>
<td>Returns a pointer to the ASCII string that represents the number in octal. A width of nonzero is assumed to be the field width for formatting. The returned value is not guaranteed to point to the beginning of the user-provided buffer.</td>
</tr>
<tr>
<td>char *hex_r (char *buf, int buflen, long num, int width)</td>
<td>Returns a pointer to the ASCII string that represents the number in hexadecimal. A width of nonzero is assumed to be the field width for formatting. The returned value is not guaranteed to point to the beginning of the user-provided buffer.</td>
</tr>
<tr>
<td>char *dec_r (char *buf, int buflen, long num, int width)</td>
<td>Returns a pointer to the ASCII string that represents the number in decimal. A width of nonzero is assumed to be the field width for formatting. The returned value is not guaranteed to point to the beginning of the user-provided buffer.</td>
</tr>
<tr>
<td>char *chr_r (char *buf, int buflen, long num, int width)</td>
<td>Returns a pointer to the ASCII string that contains character chr. If the width is nonzero, the string contains width blanks followed by chr. The returned value is not guaranteed to point to the beginning of the user-provided buffer.</td>
</tr>
<tr>
<td>char *form_r (char *buf, int buflen, long num, int width)</td>
<td>Returns a pointer of the string formatted by sprintf, using the format string format and any remaining arguments. The buffer must have sufficient space to contain the formatted string.</td>
</tr>
</tbody>
</table>

**Note** – The public conversion routines of the istream library (oct, hex, dec, chr, and form) that are present to ensure compatibility with an earlier version of libc are not MT-safe.

#### 11.4.1.2 Compiling and Linking With the MT-Safe libc Library

When you build an application that uses the istream classes of the libc library to run in a multithreaded environment, compile and link the source code of the application using the -mt option. This option passes -D_REENTRANT to the preprocessor and -lthread to the linker.
Note – Use -mt (rather than -lthread) to link with libC and libthread. This option ensures proper linking order of the libraries. Using -lthread improperly could cause your application to work incorrectly.

Single-threaded applications that use iostream classes do not require special compiler or linker options. By default, the compiler links with the libC library.

### 11.4.1.3 MT-Safe iostream Restrictions

The restricted definition of MT-safety for the iostream library means that a number of programming idioms used with iostream are unsafe in a multithreaded environment using shared iostream objects.

**Checking Error State**

To be MT-safe, error checking must occur in a critical region with the I/O operation that causes the error. The following example illustrates how to check for errors:

**EXAMPLE 11-1 Checking Error State**

```cpp
#include <iostream.h>
enum iostate {IOok, IOeof, IOfail};

iostate read_number(istream& istr, int& num)
{
    stream_locker sl(istr, stream_locker::lock_now);

    istr >> num;
    if (istr.eof()) return IOeof;
    if (istr.fail()) return IOfail;
    return IOok;
}
```

In this example, the constructor of the stream_locker object sl locks the istream object istr. The destructor of sl, called at the termination of read_number, unlocks istr.

**Obtaining Characters Extracted by Last Unformatted Input Operation**

To be MT-safe, the gcount function must be called within a thread that has exclusive use of the istream object for the period that includes the execution of the last input operation and gcount call. The following example shows a call to gcount:
EXAMPLE 11–2 Calling gcount

```cpp
#include <iostream.h>
#include <rlocks.h>
void fetch_line(istream& istr, char* line, int& linecount)
{
    stream_locker sl(istr, stream_locker::lock_defer);

    sl.lock(); // lock the stream istr
    istr >> line;
    linecount = istr.gcount();
    sl.unlock(); // unlock istr

    ...
}
```

In this example, the `lock` and `unlock` member functions of class `stream_locker` define a mutual exclusion region in the program.

**User-Defined I/O Operations**

To be MT-safe, I/O operations defined for a user-defined type that involve a specific ordering of separate operations must be locked to define a critical region. The following example shows a user-defined I/O operation:

EXAMPLE 11–3 User-Defined I/O Operations

```cpp
#include <rlocks.h>
#include <iostream.h>
class mystream: public istream {

    // other definitions...
    int getRecord(char* name, int& id, float& gpa);
};

int mystream::getRecord(char* name, int& id, float& gpa)
{
    stream_locker sl(this, stream_locker::lock_now);

    *this >> name;
    *this >> id;
    *this >> gpa;

    return this->fail() == 0;
}
```
11.4.1.4 Reducing Performance Overhead of MT-Safe Classes

Using the MT-safe classes in this version of the stdc library results in some amount of performance overhead, even in a single-threaded application; however, if you use the unsafe classes of stdc, this overhead can be avoided.

The scope resolution operator can be used to execute member functions of the base unsafe classes; for example:

```cpp
cout.unsafe_ostream::put('4');

cin.unsafe_istream::read(buf, len);
```

**Note** - The unsafe classes cannot be safely used in multithreaded applications.

Instead of using unsafe classes, you can make the cout and cin objects unsafe and then use the normal operations. A slight performance deterioration results. The following example shows how to use unsafe cout and cin:

**EXAMPLE 11-4 Disabling MT-Safety**

```cpp
#include <iostream.h>

// disable mt-safety
cout.set_safe_flag(stream_MT::unsafe_object);

// disable mt-safety
cin.set_safe_flag(stream_MT::unsafe_object);

cout.put("4");

cin.read(buf, len);
```

When an iostream object is MT-safe, mutex locking is provided to protect the object's member variables. This locking adds unnecessary overhead to an application that only executes in a single-threaded environment. To improve performance, you can dynamically switch an iostream object to and from MT-safety. The following example makes an iostream object MT-unsafe:

**EXAMPLE 11-5 Switching to MT-Unsafe**

```cpp
fs.set_safe_flag(stream_MT::unsafe_object); // disable MT-safety

.... do various i/o operations
```

You can safely use an MT-unsafe stream in code where an iostream is not shared by threads; for example, in a program that has only one thread, or in a program where each iostream is private to a thread.
If you explicitly insert synchronization into the program, you can also safely use MT-unsafe `iostreams` in an environment where an `iostream` is shared by threads. The following example illustrates the technique:

**EXAMPLE 11–6  Using Synchronization With MT-Unsafe Objects**

```cpp
generic_lock();
fs.set_safe_flag(stream_MT::unsafe_object);
... do various i/o operations
generic_unlock();
```

where the `generic_lock` and `generic_unlock` functions can be any synchronization mechanism that uses such primitives as mutex, semaphores, or reader/writer locks.

---

**Note** – The `stream_locker` class provided by the libC library is the preferred mechanism for this purpose.

See “11.4.5 Object Locks” on page 130 for more information.

### 11.4.2 Interface Changes to the iostream Library

This section describes the interface changes made to the `iostream` library to make it MT-Safe.

#### 11.4.2.1 The New Classes

The following table lists the new classes added to the libC interfaces.

**EXAMPLE 11–7  New Classes**

```cpp
stream_MT
stream_locker
unsafe_ios
unsafe_iostream
unsafe_ostream
unsafe_iostream
unsafe_fstreambase
unsafe_strstreambase
```

#### 11.4.2.2 The New Class Hierarchy

The following table lists the new class hierarchy added to the `iostream` interfaces.
11.4 Using Classic iostreams in a Multithreading Environment

**Example 11-8  New Class Hierarchy**

```cpp
class streambuf: public stream_MT {...};
class unsafe_ios {...};
class ios: virtual public unsafe_ios, public stream_MT {...};
class unsafe_fstreambase: virtual public unsafe_ios {...};
class fstreambase: virtual public ios, public unsafe_fstreambase {...};
class unsafe_strstreambase: virtual public unsafe_ios {...};
class strstreambase: virtual public ios, public unsafe_strstreambase {...};
class unsafe_istream: virtual public unsafe_ios {...};
class unsafe_ostream: virtual public unsafe_ios {...};
class istream: virtual public ios, public unsafe_istream {...};
class ostream: virtual public ios, public unsafe_ostream {...};
class unsafe_iostream: public unsafe_istream, public unsafe_ostream {...};
```

**11.4.2.3 The New Functions**

The following table lists the new functions added to the iostream interfaces.

**Example 11-9  New Functions**

```cpp
class streambuf {
public:
  int sgetc_unlocked();
  void sgetn_unlocked(char *, int);
  int snextc_unlocked();
  int sbumpc_unlocked();
  void stossc_unlocked();
  int in_avail_unlocked();
  int sputbackc_unlocked(char);
  int sputc_unlocked(int);
  int sputn_unlocked(const char *, int);
  int out_waiting_unlocked();
protected:
  char* base_unlocked();
  char* ebuf_unlocked();
  int blen_unlocked();
  char* pbase_unlocked();
  char* eback_unlocked();
  char* gptr_unlocked();
  char* egptr_unlocked();
  char* pptr_unlocked();
  void setp_unlocked(char*, char*);
  void setg_unlocked(char*, char*, char*);
  void pbump_unlocked(int);
  void gbump_unlocked(int);
  void setb_unlocked(char*, char*, int);
```
EXAMPLE 11–9  New Functions  (Continued)

    int unbuffered_unlocked();
    char *eptr_unlocked();
    void unbuffered_unlocked(int);
    int allocate_unlocked(int);
    
    class filebuf: public streambuf {
    public:
     int is_open_unlocked();
     filebuf* close_unlocked();
     filebuf* open_unlocked(const char*, int, int =
     filebuf::openprot);
     filebuf* attach_unlocked(int);
    
    class strstreambuf: public streambuf {
    public:
     int freeze_unlocked();
     char* str_unlocked();
    
    unsafe_ostream& endl(unsafe_ostream&);
    unsafe_ostream& ends(unsafe_ostream&);
    unsafe_ostream& flush(unsafe_ostream&);
    unsafe_istream& ws(unsafe_istream&);
    unsafe_ios& dec(unsafe_ios&);
    unsafe_ios& hex(unsafe_ios&);
    unsafe_ios& oct(unsafe_ios&);
    
    char* dec_r (char* buf, int buflen, long num, int width)
    char* hex_r (char* buf, int buflen, long num, int width)
    char* oct_r (char* buf, int buflen, long num, int width)
    char* chr_r (char* buf, int buflen, long chr, int width)
    char* str_r (char* buf, int buflen, const char* format, int width = 0);
    char* form_r (char* buf, int buflen, const char* format,...)

11.4.3  Global and Static Data

Global and static data in a multithreaded application are not safely shared among threads. Although threads execute independently, they share access to global and static objects within the process. If one thread modifies such a shared object, all the other threads within the process observe the change, making it difficult to maintain state over time. In C++, class objects
(instances of a class) maintain state by the values in their member variables. If a class object is shared, it is vulnerable to changes made by other threads.

When a multithreaded application uses the iostream library and includes iostream.h, the standard streams—cout, cin, cerr, and clog— are, by default, defined as global shared objects. Since the iostream library is MT-safe, it protects the state of its shared objects from access or change by another thread while a member function of an iostream object is executing. However, the scope of MT-safety for an object is confined to the period in which the object's public member function is executing. For example,

```cpp
int c;
cin.get(c);
```

gets the next character in the get buffer and updates the buffer pointer in ThreadA. However, if the next instruction in ThreadA is another get call, the libc library does not guarantee to return the next character in the sequence. It is not guaranteed because, for example, ThreadB may have also executed the get call in the intervening period between the two get calls made in ThreadA.

See “11.4.5 Object Locks” on page 130 for strategies for dealing with the problems of shared objects and multithreading.

11.4.4 Sequence Execution

Frequently, when iostream objects are used, a sequence of I/O operations must be MT-safe. For example, the code:

```cpp
cout << " Error message:" << errstring[err_number] << "\n";
```

involves the execution of three member functions of the cout stream object. Since cout is a shared object, the sequence must be executed atomically as a critical section to work correctly in a multithreaded environment. To perform a sequence of operations on an iostream class object atomically, you must use some form of locking.

The libc library now provides the stream_locker class for locking operations on an iostream object. See “11.4.5 Object Locks” on page 130 for information about the stream_locker class.

11.4.5 Object Locks

The simplest strategy for dealing with the problems of shared objects and multithreading is to avoid the issue by ensuring that iostream objects are local to a thread. For example,

- Declare objects locally within a thread's entry function.
- Declare objects in thread-specific data. (For information on how to use thread specific data, see the thr_keycreate(3T) man page.)
Dedicate a stream object to a particular thread. The object thread is private by convention. However, in many cases, such as default shared standard stream objects, it is not possible to make the objects local to a thread, and an alternative strategy is required.

To perform a sequence of operations on an iostream class object atomically, you must use some form of locking. Locking adds some overhead even to a single-threaded application. The decision whether to add locking or make iostream objects private to a thread depends on the thread model chosen for the application: Are the threads to be independent or cooperating?

- If each independent thread is to produce or consume data using its own iostream object, the iostream objects are private to their respective threads and locking is not required.
- If the threads are to cooperate (that is, they are to share the same iostream object), then access to the shared object must be synchronized and some form of locking must be used to make sequential operations atomic.

11.4.5.1 Class stream_locker

The iostream library provides the stream_locker class for locking a series of operations on an iostream object. You can, therefore, minimize the performance overhead incurred by dynamically enabling or disabling locking in iostream objects.

Objects of class stream_locker can be used to make a sequence of operations on a stream object atomic. For example, the code shown in the example below seeks to find a position in a file and reads the next block of data.

**EXAMPLE 11–10  Example of Using Locking Operations**

```cpp
#include <fstream.h>
#include <rlocks.h>

void lock_example (fstream& fs)
{
    const int len = 128;
    char buf[len];
    int offset = 48;
    stream_locker s_lock(fs, stream_locker::lock_now);
    .....
    fs.seekg(offset, ios::beg);
    fs.read(buf, len);
}
```

In this example, the constructor for the stream_locker object defines the beginning of a mutual exclusion region in which only one thread can execute at a time. The destructor, called after the return from the function, defines the end of the mutual exclusion region. The stream_locker
object ensures that both the seek to a particular offset in a file and the read from the file are performed together, atomically, and that ThreadB cannot change the file offset before the original ThreadA reads the file.

An alternative way to use a stream_locker object is to explicitly define the mutual exclusion region. In the following example, to make the I/O operation and subsequent error checking atomic, lock and unlock member function calls of a std::stream_locker object are used.

**EXAMPLE 11–11** Making I/O Operation and Error Checking Atomic

```cpp
{
...
stream_locker file_lck(openfile_stream,

    stream_locker::lock_defer);

...
file_lck.lock(); // lock openfile_stream
openfile_stream << "Value: " << int_value << "\n";
if(!openfile_stream) {
    file_error("Output of value failed\n");
    return;
}
file_lck.unlock(); // unlock openfile_stream
}
```

For more information, see the stream_locker(3C) man page.

### 11.4.6 MT-Safe Classes

You can extend or specialize the functionality of the `iostream` classes by deriving new classes. If objects instantiated from the derived classes will be used in a multithreaded environment, the classes must be MT-safe.

Considerations when deriving MT-safe classes include:

- Making a class object MT-safe by protecting the internal state of the object from multiple-thread modification. To do this, serialize access to member variables in public and protected member functions with mutex locks.
- Making a sequence of calls to member functions of an MT-safe base class atomic, using a std::stream_locker object.
- Avoiding locking overhead by using the _unlocked member functions of std::streambuf within critical regions defined by std::stream_locker objects.
- Locking the public virtual functions of class std::streambuf in case the functions are called directly by an application. These functions are: `xgetn`, `underflow`, `pbackfail`, `xputn`, `overflow`, `seekoff`, and `seekpos`. 
- Extending the formatting state of an ios object by using the member functions iword and pword in class ios. However, a problem can occur if more than one thread is sharing the same index to an iword or pword function. To make the threads MT-safe, use an appropriate locking scheme.
- Locking member functions that return the value of a member variable greater in size than a char.

### 11.4.7 Object Destruction

Before an iostream object that is shared by several threads is deleted, the main thread must verify that the subthreads are finished with the shared object. The following example shows how to safely destroy a shared object.

**EXAMPLE 11–12  Destroying a Shared Object**

```cpp
#include <fstream.h>
#include <thread.h>
fstream* fp;

void *process_rtn(void*)
{
    // body of sub-threads which uses fp...
}

void multi_process(const char* filename, int numthreads)
{
    fp = new fstream(filename, ios::in); // create fstream object
    // before creating threads.
    // create threads
    for (int i=0; i<numthreads; i++)
        thr_create(0, STACKSIZE, process_rtn, 0, 0, 0);
    ...
    // wait for threads to finish
    for (int i=0; i<numthreads; i++)
        thr_join(0, 0, 0);

    delete fp; // delete fstream object after
    fp = NULL; // all threads have completed.
}
```

### 11.4.8 An Example Application

The following code provides an example of a multiply-threaded application that uses iostream objects from the libc library in an MT-safe way.
The example application creates up to 255 threads. Each thread reads a different input file, one line at a time, and outputs the line to an output file, using the standard output stream, cout. The output file, which is shared by all threads, is tagged with a value that indicates which thread performed the output operation.

**EXAMPLE 11-13 Using iostream Objects in an MT-Safe Way**

```cpp
// create tagged thread data
// the output file is of the form:
//  <tag><string of data>
// where tag is an integer value in a unsigned char.
// Allows up to 255 threads to be run in this application
// <string of data> is any printable characters
// Because tag is an integer value written as char,
// you need to use od to look at the output file, suggest:
//   od -c out.file |more

#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <thread.h>

struct thread_args {
    char* filename;
    int thread_tag;
};

const int thread_bufsize = 256;

// entry routine for each thread
void* ThreadDuties(void* v) {
    // obtain arguments for this thread
    thread_args* tt = (thread_args*)v;
    char ibuf[thread_bufsize];
    // open thread input file
    ifstream instr(tt->filename);
    stream_locker lockout(cout, stream_locker::lock_defer);
    while(1) {
        // read a line at a time
        instr.getline(ibuf, thread_bufsize - 1, '\n');
        if(instr.eof())
            break;
        // lock cout stream so the i/o operation is atomic
        lockout.lock();
        // tag line and send to cout
        cout << (unsigned char)tt->thread_tag << ibuf << "\n";
        lockout.unlock();
    }
}
```
```c
EXAMPLE 11–13  Using iostream Objects in an MT-Safe Way  (Continued)

}  // argv: 1+ list of filenames per thread
if(argc < 2) {
    cout << "usage: " << argv[0] << " <files..>\n";
    exit(1);
}  // array of thread_ids
    thread_t created_threads[thread_bufsize];
    // array of arguments to thread entry routine
    thread_args thr_args[thread_bufsize];
    int i;
    for(i = 0; i < num_threads; i++) {
        thr_args[i].filename = argv[1 + i];
        // assign a tag to a thread - a value less than 256
        thr_args[i].thread_tag = total_tags++;
        // create threads
        thr_create(0, 0, ThreadDuties, &thr_args[i],
                   THR_SUSPENDED, &created_threads[i]);
    }
    for(i = 0; i < num_threads; i++) {
        thr_continue(created_threads[i]);
    }
    for(i = 0; i < num_threads; i++) {
        thr_join(created_threads[i], 0, 0);
    }
    return 0;
}
```
PART III

Libraries
Using Libraries

Libraries provide a way to share code among several applications and a way to reduce the complexity of very large applications. The C++ compiler gives you access to a variety of libraries. This chapter explains how to use these libraries.

12.1 The C Libraries

The Solaris operating system comes with several libraries installed in /usr/lib. Most of these libraries have a C interface. Of these, the libc and libm libraries are linked by the CC driver by default. The library libthread is linked if you use the -mt option. To link any other system library, use the appropriate -l option at link time. For example, to link the libdemangle library, pass -ldemangle on the CC command line at link time:

example% CC text.c -ldemangle

The C++ compiler has its own runtime support libraries. All C++ applications are linked to these libraries by the CC driver. The C++ compiler also comes with several other useful libraries, as explained in the following section.

12.2 Libraries Provided With the C++ Compiler

Several libraries are shipped with the C++ compiler. Some of these libraries are available only in compatibility mode (-compat=4), some are available only in the standard mode (-compat=5), and some are available in both modes. The libgc and libdemangle libraries have a C interface and can be linked to an application in either mode.

The following table lists the libraries that are shipped with the C++ compiler and the modes in which they are available.
### TABLE 12-1 Libraries Shipped With the C++ Compiler

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
<th>Available Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>libstlport</td>
<td>STLport implementation of the standard library.</td>
<td>–compat=5</td>
</tr>
<tr>
<td>libstlport_dbg</td>
<td>STLport library for debug mode</td>
<td>–compat=5</td>
</tr>
<tr>
<td>libcrun</td>
<td>C++ runtime</td>
<td>–compat=5</td>
</tr>
<tr>
<td>libcstd</td>
<td>C++ standard library</td>
<td>–compat=5</td>
</tr>
<tr>
<td>libiostream</td>
<td>Classic iostreams</td>
<td>–compat=5</td>
</tr>
<tr>
<td>libc</td>
<td>C++ runtime, classic iostreams</td>
<td>–compat=4</td>
</tr>
<tr>
<td>libcsunimath</td>
<td>Supports the -xia option</td>
<td>–compat=5</td>
</tr>
<tr>
<td>libcomplex</td>
<td>complex-number library</td>
<td>–compat=4</td>
</tr>
<tr>
<td>librwtool</td>
<td>Tools.h++7</td>
<td>–compat=4, –compat=5</td>
</tr>
<tr>
<td>librwtool_dbg</td>
<td>Debug-enabled Tools.h++7</td>
<td>–compat=4, –compat=5</td>
</tr>
<tr>
<td>libgc</td>
<td>Garbage collection</td>
<td>C interface</td>
</tr>
<tr>
<td>libdemangle</td>
<td>Demangling</td>
<td>C interface</td>
</tr>
</tbody>
</table>

**Note** – Do not redefine or modify any of the configuration macros for STLport, Rogue Wave or Sun Microsystems C++ libraries. The libraries are configured and built in a way that works with the C++ compiler. libcstd and Tool.h++ are configured to inter-operate so modifying the configuration macros results in programs that will not compile, will not link, or do not run properly.

#### 12.2.1 C++ Library Descriptions

A brief description of each of these libraries follows.

- **libCrun**: This library contains the runtime support needed by the compiler in the standard mode (–compat=5). It provides support for `new/delete`, exceptions, and RTTI.
- **libcstd**: This is the C++ standard library. In particular, it includes `iostreams`. If you have existing sources that use the classic `iostreams` and you want to make use of the standard `iostreams`, you have to modify your sources to conform to the new interface. See the C++ Standard Library Reference online manual for details.

If your compiler software is not installed in the `/opt` directory, ask your system administrator for the equivalent path on your system.
libiostream: This is the classic iostreams library built with –compat=5. If you have existing sources that use the classic iostreams and you want to compile these sources with the standard mode (–compat=5), you can use libiostream without modifying your sources. Use–library=iostream to get this library.

**Note** – Much of the standard library depends on using standard iostreams. Using classic iostreams in the same program can cause problems.

LibC: This is the library needed in compatibility mode (–compat=4). It contains the C++ runtime support as well as the classic iostreams.

libcomplex: This library provides complex arithmetic in compatibility mode (–compat=4). In the standard mode, the complex arithmetic functionality is available in libCstd.

libstlport: This is the STLport implementation of the C++ standard library. You can use this library instead of the default libCstd by specifying the option -library=stlport4. However, you cannot use libstlport and libCstd in the same program. You must compile and link everything, including imported libraries, using one or the other exclusively.

librwtool (Tools.h++): Tools.h++ is a C++ foundation class library from RogueWave. Version 7 is provided. This library is obsolete and use of the library is deprecated in new code. It is provided to accommodate programs written for C++ 4.2 that used RW Tools.h++.

libgc: This library is used in deployment mode or garbage collection mode. Simply linking with the libgc library automatically and permanently fixes a program’s memory leaks. When you link your program with the libgc library, you can program without calling free or delete while otherwise programming normally. The garbage collection library has a dependency on the dynamic load library so specify -lgc and -ldl when you link your program.

Additional information can be found in the gcFixPrematureFrees(3) and gcInitialize(3) man pages.

libdemangle: This library is used for demangling C++ mangled names.

### 12.2.2 Accessing the C++ Library Man Pages

The man pages associated with the libraries described in this section are located in sections 1, 3, 3C++, and 3cc4.

To access man pages for the C++ libraries, type:

```
example% man library-name
```

To access man pages for version 4.2 of the C++ libraries, type:

```
example% man -s 3CC4 library-name
```
12.2.3 Default C++ Libraries

Some of the C++ libraries are linked by default by the CC driver, while others need to be linked explicitly. In the standard mode, the following libraries are linked by default by the CC driver:

-1Cstd -1Crun -1m -1c

In compatibility mode (-compat), the following libraries are linked by default:

-1C -1m -1c

See “A.2.50 -library=[,...]” on page 252 for more information.

12.3 Related Library Options

The CC driver provides several options to help you use libraries.

- Use the -l option to specify a library to be linked.
- Use the -L option to specify a directory to be searched for the library.
- Use the -mt option compile and link multithreaded code.
- Use the -xia option to link the interval arithmetic libraries.
- Use the -xlang option to link Fortran or C99 runtime libraries.
- Use the -library option to specify the following libraries that are shipped with the Sun C++ compiler:
  - libCrun
  - libCstd
  - libiostream
  - libC
  - libcomplex
  - libstlport, libstlport_dbg
  - librwwtool, librwwtool_dbg
  - libgc

Note  To use the classic-iostreams form of librwwtool, use the -library=rwwtools7 option. To use the standard-iostreams form of librwwtool, use the -library=rwwtools7_std option.

A library that is specified using both -library and -staticlib options will be linked statically. Some examples:
The following command links the classic-iostreams form of Tools.h++ version 7 and libiostream libraries dynamically.

```
example% CC test.cc -library=rwtools7,iostream
```

■ The following command links the libgc library statically.

```
example% CC test.cc -library=gc -staticlib=gc
```

■ The following command compiles test.cc in compatibility mode and links libC statically. Because libC is linked by default in compatibility mode, you are not required to specify this library using the -library option.

```
example% CC test.cc -compat=4 -staticlib=libC
```

■ The following command excludes the libraries libCrun and libCstd, which would otherwise be included by default.

```
example% CC test.cc -library=no%Crun,no%Cstd
```

By default, CC links various sets of system libraries depending on the command line options. If you specify -xnolib (or -nolib), CC links only those libraries that are specified explicitly with the -l option on the command line. (When -xnolib or -nolib is used, the -library option is ignored, if present.)

The -R option allows you to build dynamic library search paths into the executable file. At execution time, the runtime linker searches these paths for the shared libraries needed by the application. The CC driver passes -R<install_directory>/lib to ld by default (if the compiler is installed in the standard location). You can use -norunpath to disable building the default path for shared libraries into the executable.

Programs built for deployment should be built with -norunpath or an -R option that that avoids looking in the compiler directory for libraries. (See "12.6 Using Shared Libraries" on page 148).

### 12.4 Using Class Libraries

Generally, two steps are involved in using a class library:

1. Include the appropriate header in your source code.
2. Link your program with the object library.

#### 12.4.1 The iostream Library

The C++ compiler provides two implementations of iostreams:
12.4 Using Class Libraries

- **Classic iostreams.** This term refers to the iostreams library shipped with the C++ 4.0, 4.0.1, 4.1, and 4.2 compilers, and earlier with the cfront-based 3.0.1 compiler. There is no standard for this library, but a lot of existing code uses it. This library is part of libC in compatibility mode and is also available in libiostream in the standard mode.

- **Standard iostreams.** This is part of the C++ standard library, libCstd, and is available only in standard mode. It is neither binary-, nor source-compatible with the classic iostreams library.

If you have existing C++ sources, your code might look like the following example, which uses classic iostreams.

```cpp
// file prog1.cc
#include <iostream.h>

int main() {
    cout << "Hello, world!" << endl;
    return 0;
}
```

The following command compiles in compatibility mode and links prog1.cc into an executable program called prog1. The classic iostream library is part of libC, which is linked by default in compatibility mode.

```
example% CC -compat prog1.cc -o prog1
```

The next example uses standard iostreams.

```cpp
// file prog2.cc
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

The following command compiles and links prog2.cc into an executable program called prog2. The program is compiled in standard mode and libCstd, which includes the standard iostream library, is linked by default.

```
example% CC prog2.cc -o prog2
```

For more information about libCstd, see “Caveats” on page 152. For more information about libiostream, see “13.3.1 Redistribution and Supported STLport Libraries” on page 169.

For a full discussion of compilation modes, see the C++ Migration Guide.
12.4.2 The complex Library

The standard library provides a templated complex library that is similar to the complex library provided with the C++ 4.2 compiler. If you compile in standard mode, you must use <complex> instead of <complex.h>. You cannot use <complex> in compatibility mode.

In compatibility mode, you must explicitly ask for the complex library when linking. In standard mode, the complex library is included in libCstd, and is linked by default.

There is no complex.h header for standard mode. In C++ 4.2, "complex" is the name of a class, but in standard C++, "complex" is the name of a template. It is not possible to provide typedefs that allow old code to work unchanged. Therefore, code written for 4.2 that uses complex numbers will need some straightforward editing to work with the standard library. For example, the following code was written for 4.2 and will compile in compatibility mode.

```cpp
// file ex1.cc (compatibility mode)
#include <iostream.h>
#include <complex.h>

int main()
{
    complex x(3,3), y(4,4);
    complex z = x * y;
    cout << "x=" << x << " y=" << y << " z=" << z << endl;
}
```

The following example compiles and links ex1.cc in compatibility mode, and then executes the program.

```
example% CC -compat ex1.cc -library=complex
example% a.out
x=(3, 3), y=(4, 4), z=(0, 24)
```

Here is ex1.cc rewritten as ex2.cc to compile in standard mode:

```cpp
// file ex2.cc (ex1.cc rewritten for standard mode)
#include <iostream>
#include <complex>
using std::complex;

int main()
{
    complex<double> x(3,3), y(4,4);
    complex<double> z = x * y;
    std::cout << "x=" << x << " y=" << y << " z=" << z << std::endl;
}
```
The following example compiles and links the rewritten `ex2.cc` in standard mode, and then executes the program.

```plaintext
% CC ex2.cc
% a.out
```

x=(3,3), y=(4,4), z=(0,24)

For more information about using the complex arithmetic library, see Table 14–4.

### 12.4.3 Linking C++ Libraries

The following table shows the compiler options for linking the C++ libraries. See "A.2.50 -library=..." on page 252 for more information.

<table>
<thead>
<tr>
<th>Library</th>
<th>Compilation Mode</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic iostream</td>
<td>-compat=4</td>
<td>None needed</td>
</tr>
<tr>
<td></td>
<td>-compat=5</td>
<td>:library=iostream</td>
</tr>
<tr>
<td>complex</td>
<td>-compat=4</td>
<td>:library=complex</td>
</tr>
<tr>
<td></td>
<td>-compat=5</td>
<td>None needed</td>
</tr>
<tr>
<td>Tools.h++ version 7</td>
<td>-compat=4</td>
<td>:library=rtools7</td>
</tr>
<tr>
<td></td>
<td>-compat=5</td>
<td>:library=rtools7,iostream</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:library=rtools7_std</td>
</tr>
<tr>
<td>Tools.h++ version 7 debug</td>
<td>-compat=4</td>
<td>:library=rtools7_dbg</td>
</tr>
<tr>
<td></td>
<td>-compat=5</td>
<td>:library=rtools7_dbg,iostream</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:library=rtools7_dbg_std</td>
</tr>
<tr>
<td>Garbage collection</td>
<td>-compat=4</td>
<td>:library=gc</td>
</tr>
<tr>
<td></td>
<td>-compat=5</td>
<td>:library=gc</td>
</tr>
<tr>
<td>STLport version 4</td>
<td>-compat=5</td>
<td>:library=stlport4</td>
</tr>
<tr>
<td>STLport version 4 debug</td>
<td>-compat=5</td>
<td>:library=stlport4_dbg</td>
</tr>
</tbody>
</table>
12.5 Statically Linking Standard Libraries

The CC driver links in shared versions of several libraries by default, including libc and libm, by passing a -l option for each of the default libraries to the linker. (See “12.2.3 Default C++ Libraries” on page 142 for the list of default libraries for compatibility mode and standard mode.)

If you want any of these default libraries to be linked statically, you can use the -l option along with the -static option to link a C++ library statically. This alternative is much easier than the one described earlier. For example:

```
example% CC test.c -static
```

In this example, the -library option is not explicitly included in the command. In this case the -library option is not necessary because the default setting for -library is Cstd, Crun in standard mode (the default mode).

Alternately, you can use the -xnolib compiler option. With the -xnolib option, the driver does not pass any -l options to ld; you must pass these options yourself. The following example shows how you would link statically with libcRUN, and dynamically with librM, and libc in the Solaris 8, or Solaris 9 operating systems:

```
example% CC test.c -xnolib -lCstd -Bsstatic -lCrun -Bdynamic -lm -lc
```

The order of the -l options is important. The -lCstd, -lCrun, and -lm options appear before -lc.

**Note** – Linking the libcRUN and libcCstd statically is not recommended. The dynamic versions in /usr/lib are built to work with the version of Solaris where they are installed.

Some CC options link to other libraries. These library links are also suppressed by -xnolib. For example, using the -mt option causes the CC driver to pass -ltthread to ld. However, if you use both -mt and -xnolib, the CC driver does not pass -ltthread to ld. See “A.2.153 –xnolib” on page 319 for more information. See Linker and Libraries Guide for more information about ld.

**Note** – Static versions of Solaris libraries in /lib and /usr/lib are no longer available. For example this attempt to link libc statically will fail:

```
CC hello.cc -xnolib -lCrun -lCstd -Bsstatic -lc
```
12.6 Using Shared Libraries

The following C++ runtime shared libraries are shipped as part of the C++ compiler:

- libCExcept.so.1 (SPARC Solaris only)
- libcomplex.so.5 (Solaris only)
- librwtool.so.2
- libstlport.so.1

On Linux, these additional libraries are shipped as part of the C++ compiler:

- libCrun.so.1
- libCstd.so.1
- libdemangle.so
- libiostream.so.1

On Solaris, those additional libraries, along with some others, are installed as part of the Solaris C++ runtime library package, SUNWlibC.

If your application uses any of the shared libraries that are shipped as part of the C++ compiler, the CC driver arranges for a runpath (refer to the -R option) pointing to the location of the library to be built into the executable. If the executable is later deployed to a different computer where the same compiler version is not installed in the same location, the required shared library will not be found.

At program start time, the library might not be found at all, or the wrong version of the library might be used, leading to incorrect program behavior. In such a case, you should ship the required libraries along with the executable, and build with a runpath that points to where they will be installed.

The article Using and Redistributing Sun Studio Libraries in an Application contains a full discussion of this topic, along with examples and can be found at http://developers.sun.com/sunstudio/documentation/techart/stdlibdistr.html

12.7 Replacing the C++ Standard Library

Replacing the standard library that is distributed with the compiler is risky, and good results are not guaranteed. The basic operation is to disable the standard headers and library supplied with the compiler, and to specify the directories where the new header files and library are found, as well as the name of the library itself.

The compiler supports the STLport implementation of the standard library. See "13.3 STLport" on page 168 for more information.
12.7.1 What Can Be Replaced

You can replace most of the standard library and its associated headers. The replaced library is libCstd, and the associated headers are the following:

\[ \text{<algorithm>} \ <\text{bitset}> \ <\text{complex}> \ <\text{deque}> \ <\text{fstream}> \ <\text{functional}> \ <\text{iomanip}> \ <\text{ios>} \ <\text{iosfwd}> \ <\text{iostream}> \ <\text{iostream}> \ <\text{iterator}> \ <\text{limits}> \ <\text{list}> \ <\text{locale}> \ <\text{map}> \ <\text{memory}> \ <\text{numeric}> \ <\text{ostream}> \ <\text{queue}> \ <\text{set}> \ <\text{sstream}> \ <\text{stack}> \ <\text{stdexcept}> \ <\text{streambuf}> \ <\text{string}> \ <\text{strstream}> \ <\text{utility}> \ <\text{valarray}> \ <\text{vector}> \]

The replaceable part of the library consists of what is loosely known as “STL”, plus the string classes, the iostream classes, and their helper classes. Because these classes and headers are interdependent, replacing just a portion of them is unlikely to work. You should replace all of the headers and all of libCstd if you replace any part.

12.7.2 What Cannot Be Replaced

The standard headers <exception>, <new>, and <typeinfo> are tied tightly to the compiler itself and to libCrun, and cannot reliably be replaced. The library libCrun contains many “helper” functions that the compiler depends on, and cannot be replaced.

The 17 standard headers inherited from C (<stdio.h>, <stdlib.h>, <string.h>, and so forth) are tied tightly to the Solaris operating system and the basic Solaris runtime library libc, and cannot reliably be replaced. The C++ versions of those headers (<cstdlib>, <cstdio>, <cstring>, and so forth) are tied tightly to the basic C versions and cannot reliably be replaced.

12.7.3 Installing the Replacement Library

To install the replacement library, you must first decide on the locations for the replacement headers and on the replacement for libCstd. For purposes of discussion, assume the headers are placed in /opt/mycstd/include and the library is placed in /opt/mycstd/lib. Assume the library is called libmyCstd.a. (It is often convenient if the library name starts with “lib”.)

12.7.4 Using the Replacement Library

On each compilation, use the -I option to point to the location where the headers are installed. In addition, use the -library=no%Cstd option to prevent finding the compiler’s own versions of the libCstd headers. For example:

\[ \text{example } \text{CC} -I /opt/mycstd/include -\text{library=no\%Cstd} \ldots \ (\text{compile}) \]

During compiling, the -library=no%Cstd option prevents searching the directory where the compiler’s own version of these headers is located.
On each program or library link, use the `-library=no%Cstd` option to prevent finding the compiler's own `libCstd`, the `-L` option to point to the directory where the replacement library is, and the `-l` option to specify the replacement library. Example:

```
example% CC -library=no%Cstd -L/opt/mycstd/lib -lmyCstd... (link)
```

Alternatively, you can use the full path name of the library directly, and omit using the `-L` and `-l` options. For example:

```
example% CC -library=no%Cstd /opt/mycstd/lib/libmyCstd.a... (link)
```

During linking, the `-library=no%Cstd` option prevents linking the compiler's own version of `libCstd`.

### 12.7.5 Standard Header Implementation

C has 17 standard headers (`<stdio.h>`, `<string.h>`, `<stdlib.h>`, and others). These headers are delivered as part of the Solaris operating system, in the directory `/usr/include`. C++ has those same headers, with the added requirement that the various declared names appear in both the global namespace and in namespace `std`. On versions of the Solaris operating system prior to version 8, the C++ compiler supplies its own versions of these headers instead of replacing those in the `/usr/include` directory.

C++ also has a second version of each of the C standard headers (`<cstdio>`, `<cstring>`, and `<cstdlib>`, and others) with the various declared names appearing only in namespace `std`. Finally, C++ adds 32 of its own standard headers (`<string>`, `<utility>`, `<iostream>`, and others).

The obvious implementation of the standard headers would use the name found in C++ source code as the name of a text file to be included. For example, the standard headers `<string>` (or `<string.h>`) would refer to a file named `string` (or `string.h`) in some directory. That obvious implementation has the following drawbacks:

- You cannot search for just header files or create a `makefile` rule for the header files if they do not have file name suffixes.
- If you have a directory or executable program named `string`, it might erroneously be found instead of the standard header file.
- On versions of the Solaris operating system prior to the Solaris 8 operating system, the default dependencies for `makefiles` when `.KEEP_STATE` is enabled can result in attempts to replace standard headers with an executable program. (A file without a suffix is assumed by default to be a program to be built.)

To solve these problems, the compiler `include` directory contains a file with the same name as the header, along with a symbolic link to it that has the unique suffix `.SUNW%CCh` (SUNW is the prefix for all compiler-related packages, `CC` is the C++ compiler, and h is the usual suffix for
header files). When you specify `<string>`, the compiler rewrites it to `<string.SUNWCCh>` and searches for that name. The suffixed name will be found only in the compiler’s own include directory. If the file so found is a symbolic link (which it normally is), the compiler dereferences the link exactly once and uses the result (`string` in this case) as the file name for error messages and debugger references. The compiler uses the suffixed name when emitting file dependency information.

The name rewriting occurs only for the two forms of the 17 standard C headers and the 32 standard C++ headers, only when they appear in angle brackets and without any path specified. If you use quotes instead of angle brackets, specify any path components, or specify some other header, no rewriting occurs.

The following table illustrates common situations.

**TABLE 12-3  Header Search Examples**

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Compiler Searches For</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;string&gt;</code></td>
<td>string.SUNWCCh</td>
<td>C++ string templates</td>
</tr>
<tr>
<td><code>&lt;cstring&gt;</code></td>
<td>cstring.SUNWCCh</td>
<td>C++ version of C string.h</td>
</tr>
<tr>
<td><code>&lt;string.h&gt;</code></td>
<td>string.h.SUNWCCh</td>
<td>C string.h</td>
</tr>
<tr>
<td><code>&lt;fcntl.h&gt;</code></td>
<td>fcntl.h</td>
<td>Not a standard C or C++ header</td>
</tr>
<tr>
<td>&quot;string&quot;</td>
<td>string</td>
<td>Double-quotation marks, not angle brackets</td>
</tr>
<tr>
<td><code>&lt;../string&gt;</code></td>
<td>../string</td>
<td>Path specified</td>
</tr>
</tbody>
</table>

If the compiler does not find `header.SUNWCCh`, the compiler restarts the search looking for the name as provided in the `#include` directive. For example, given the directive `#include `<string>`, the compiler attempts to find a file named `string.SUNWCCh`. If that search fails, the compiler looks for a file named `string`.

### 12.7.5.1 Replacing Standard C++ Headers

Because of the search algorithm described in “12.7.5 Standard Header Implementation” on page 150, you do not need to supply SUNWCCh versions of the replacement headers described in “12.7.3 Installing the Replacement Library” on page 149. But you might run into some of the described problems. If so, the recommended solution is to add symbolic links having the suffix `.SUNWCCh` for each of the unsuffixed headers. That is, for file `utility`, you would run the command

```
exampel% ln -s utility utility.SUNWCCh
```

When the compiler looks first for `utility.SUNWCCh`, it will find it, and not be confused by any other file or directory called `utility`. 
12.7.5.2 Replacing Standard C Headers

Replacing the standard C headers is not supported. If you nevertheless wish to provide your own versions of standard headers, the recommended procedure is as follows:

- Put all the replacement headers in one directory.
- Create a `.SUNWCCh` symbolic link to each of the replacement headers in that directory.
- Cause the directory that contains the replacement headers to be searched by using the `-I` directives on each invocation of the compiler.

For example, suppose you have replacements for `<stdio.h>` and `<cstdio>`. Put the files `stdio.h` and `cstdio` in directory `/myproject/myhdr`. In that directory, run these commands:

```bash
example% ln -s stdio.h stdio.h.SUNWCCh
example% ln -scstdiocstdio.SUNWCCh
```

Use the option `-I/myproject/mydir` on every compilation.

**Caveats:**

- If you replace any C headers, you must replace them in pairs. For example, if you replace `<time.h>`, you should also replace `<ctime>`.

- Replacement headers must have the same effects as the versions being replaced. That is, the various runtime libraries such as `libCrun`, `libC`, `libCstd`, `libc`, and `librwtool` are built using the definitions in the standard headers. If your replacements do not match, your program is unlikely to work.
Using The C++ Standard Library

When compiling in default (standard) mode, the compiler has access to the complete library specified by the C++ standard. The library components include what is informally known as the Standard Template Library (STL), as well as the following components.

- string classes
- numeric classes
- the standard version of stream I/O classes
- basic memory allocation
- exception classes
- run-time type information

The term STL does not have a formal definition, but is usually understood to include containers, iterators, and algorithms. The following subset of the standard library headers can be thought of as comprising the STL.

- `<algorithm>`
- `<deque>`
- `<iterator>`
- `<list>`
- `<map>`
- `<memory>`
- `<queue>`
- `<set>`
- `<stack>`
- `<utility>`
- `<vector>`

The C++ standard library (libCstd) is based on the RogueWave™ Standard C++ Library, Version 2. This library is available only for the default mode (`-compat=5`) of the compiler and is not supported with use of the `-compat=[4]` option.
The C++ compiler also supports STLport’s Standard Library implementation version 4.5.3. **libCstd** is still the default library, but STLport’s product is available as an alternative. See “13.3 STLport” on page 168 for more information.

If you need to use your own version of the C++ standard library instead of one of the versions that is supplied with the compiler, you can do so by specifying the `-library=no%Cstd` option. Replacing the standard library that is distributed with the compiler is risky, and good results are not guaranteed. For more information, see “12.7 Replacing the C++ Standard Library” on page 148.

For details about the standard library, see the *Standard C++ Library User’s Guide* and the *Standard C++ Class Library Reference*.

### 13.1 C++ Standard Library Header Files

Table 13–1 lists the headers for the complete standard library along with a brief description of each.

<table>
<thead>
<tr>
<th>Header File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;algorithm&gt;</code></td>
<td>Standard algorithms that operate on containers</td>
</tr>
<tr>
<td><code>&lt;bitset&gt;</code></td>
<td>Fixed-size sequences of bits</td>
</tr>
<tr>
<td><code>&lt;complex&gt;</code></td>
<td>The numeric type representing complex numbers</td>
</tr>
<tr>
<td><code>&lt;deque&gt;</code></td>
<td>Sequences supporting addition and removal at each end</td>
</tr>
<tr>
<td><code>&lt;exception&gt;</code></td>
<td>Predefined exception classes</td>
</tr>
<tr>
<td><code>&lt;fstream&gt;</code></td>
<td>Stream I/O on files</td>
</tr>
<tr>
<td><code>&lt;functional&gt;</code></td>
<td>Function objects</td>
</tr>
<tr>
<td><code>&lt;iomanip&gt;</code></td>
<td>iostream manipulators</td>
</tr>
<tr>
<td><code>&lt;ios&gt;</code></td>
<td>iostream base classes</td>
</tr>
<tr>
<td><code>&lt;iosfwd&gt;</code></td>
<td>Forward declarations of iostream classes</td>
</tr>
<tr>
<td><code>&lt;iostream&gt;</code></td>
<td>Basic stream I/O functionality</td>
</tr>
<tr>
<td><code>&lt;istream&gt;</code></td>
<td>Input I/O streams</td>
</tr>
<tr>
<td><code>&lt;iterator&gt;</code></td>
<td>Class for traversing a sequence</td>
</tr>
<tr>
<td><code>&lt;limits&gt;</code></td>
<td>Properties of numeric types</td>
</tr>
</tbody>
</table>
### 13.2 C++ Standard Library Man Pages

Table 13–2 lists the documentation available for each of the components of the standard library.

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td>Generic algorithms for performing various operations on containers and sequences</td>
</tr>
<tr>
<td>Associative Containers</td>
<td>Ordered containers</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Bidirectional_Iterators</td>
<td>An iterator that can both read and write and can traverse a container in both directions</td>
</tr>
<tr>
<td>Containers</td>
<td>A standard template library (STL) collection</td>
</tr>
<tr>
<td>Forward_Iterators</td>
<td>A forward-moving iterator that can both read and write</td>
</tr>
<tr>
<td>Function_Objects</td>
<td>Object with an <code>operator()</code> defined</td>
</tr>
<tr>
<td>Heap_Operations</td>
<td>See entries for <code>make_heap</code>, <code>pop_heap</code>, <code>push_heap</code> and <code>sort_heap</code></td>
</tr>
<tr>
<td>Input_Iterators</td>
<td>A read-only, forward moving iterator</td>
</tr>
<tr>
<td>Insert_Iterators</td>
<td>An iterator adaptor that allows an iterator to insert into a container rather than overwrite elements in the container</td>
</tr>
<tr>
<td>Iterators</td>
<td>Pointer generalizations for traversal and modification of collections</td>
</tr>
<tr>
<td>Negators</td>
<td>Function adaptors and function objects used to reverse the sense of predicate function objects</td>
</tr>
<tr>
<td>Operators</td>
<td>Operators for the C++ Standard Template Library Output</td>
</tr>
<tr>
<td>Output_Iterators</td>
<td>A write-only, forward moving iterator</td>
</tr>
<tr>
<td>Predicates</td>
<td>A function or a function object that returns a boolean (true/false) value or an integer value</td>
</tr>
<tr>
<td>Random_Access_Iterators</td>
<td>An iterator that reads, writes, and allows random access to a container</td>
</tr>
<tr>
<td>Sequences</td>
<td>A container that organizes a set of sequences</td>
</tr>
<tr>
<td>Stream_Iterators</td>
<td>Includes iterator capabilities for ostream and istreams that allow generic algorithms to be used directly on streams</td>
</tr>
<tr>
<td>__distance_type</td>
<td>Determines the type of distance used by an iterator—obsolete</td>
</tr>
<tr>
<td>__iterator_category</td>
<td>Determines the category to which an iterator belongs—obsolete</td>
</tr>
<tr>
<td>__reverse_bi_iterator</td>
<td>An iterator that traverses a collection backwards</td>
</tr>
<tr>
<td>accumulate</td>
<td>Accumulates all elements within a range into a single value</td>
</tr>
<tr>
<td>adjacent_difference</td>
<td>Outputs a sequence of the differences between each adjacent pair of elements in a range</td>
</tr>
<tr>
<td>adjacent_find</td>
<td>Find the first adjacent pair of elements in a sequence that are equivalent</td>
</tr>
<tr>
<td>advance</td>
<td>Moves an iterator forward or backward (if available) by a certain distance</td>
</tr>
</tbody>
</table>
### Man Pages for C++ Standard Library (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocator</td>
<td>The default allocator object for storage management in Standard Library containers</td>
</tr>
<tr>
<td>auto_ptr</td>
<td>A simple, smart pointer class</td>
</tr>
<tr>
<td>back_insert_iterator</td>
<td>An insert iterator used to insert items at the end of a collection</td>
</tr>
<tr>
<td>back_inserter</td>
<td>An insert iterator used to insert items at the end of a collection</td>
</tr>
<tr>
<td>basic_filebuf</td>
<td>Class that associates the input or output sequence with a file</td>
</tr>
<tr>
<td>basic_fstream</td>
<td>Supports reading and writing of named files or devices associated with a file descriptor</td>
</tr>
<tr>
<td>basic_ifstream</td>
<td>Supports reading from named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td>basic_ios</td>
<td>A base class that includes the common functions required by all streams</td>
</tr>
<tr>
<td>basic_iostream</td>
<td>Assists in formatting and interpreting sequences of characters controlled by a stream buffer</td>
</tr>
<tr>
<td>basic_istream</td>
<td>Assists in reading and interpreting input from sequences controlled by a stream buffer</td>
</tr>
<tr>
<td>basic_istringstream</td>
<td>Supports reading objects of class basic_string&lt;charT,traits,Allocator&gt; from an array in memory</td>
</tr>
<tr>
<td>basic_ofstream</td>
<td>Supports writing into named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td>basic_ostream</td>
<td>Assists in formatting and writing output to sequences controlled by a stream buffer</td>
</tr>
<tr>
<td>basic_ostringstream</td>
<td>Supports writing objects of class basic_string&lt;charT,traits,Allocator&gt;</td>
</tr>
<tr>
<td>basic_streambuf</td>
<td>Abstract base class for deriving various stream buffers to facilitate control of character sequences</td>
</tr>
<tr>
<td>basic_string</td>
<td>A templatized class for handling sequences of character-like entities</td>
</tr>
<tr>
<td>basic_stringbuf</td>
<td>Associates the input or output sequence with a sequence of arbitrary characters</td>
</tr>
<tr>
<td>basic_stringstream</td>
<td>Supports writing and reading objects of class basic_string&lt;charT,traits,Allocator&gt; to or from an array in memory</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>binary_function</td>
<td>Base class for creating binary function objects</td>
</tr>
<tr>
<td>binary_negate</td>
<td>A function object that returns the complement of the result of its</td>
</tr>
<tr>
<td></td>
<td>binary predicate</td>
</tr>
<tr>
<td>binary_search</td>
<td>Performs a binary search for a value on a container</td>
</tr>
<tr>
<td>bind1st</td>
<td>Templated utilities to bind values to function objects</td>
</tr>
<tr>
<td>bind2nd</td>
<td>Templated utilities to bind values to function objects</td>
</tr>
<tr>
<td>binder1st</td>
<td>Templated utilities to bind values to function objects</td>
</tr>
<tr>
<td>binder2nd</td>
<td>Templated utilities to bind values to function objects</td>
</tr>
<tr>
<td>bitset</td>
<td>A template class and related functions for storing and manipulating</td>
</tr>
<tr>
<td></td>
<td>fixed-size sequences of bits</td>
</tr>
<tr>
<td>cerr</td>
<td>Controls output to an unbuffered stream buffer associated with</td>
</tr>
<tr>
<td></td>
<td>the object stderr declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>char_traits</td>
<td>A traits class with types and operations for the basic_string</td>
</tr>
<tr>
<td></td>
<td>container and iostream classes</td>
</tr>
<tr>
<td>cin</td>
<td>Controls input from a stream buffer associated with the object</td>
</tr>
<tr>
<td></td>
<td>stdin declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>clog</td>
<td>Controls output to a stream buffer associated with the object</td>
</tr>
<tr>
<td></td>
<td>stderr declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>codecvt</td>
<td>A code conversion facet</td>
</tr>
<tr>
<td>codecvt_byname</td>
<td>A facet that includes code set conversion classification facilities</td>
</tr>
<tr>
<td></td>
<td>based on the named locales</td>
</tr>
<tr>
<td>collate</td>
<td>A string collation, comparison, and hashing facet</td>
</tr>
<tr>
<td>collate_byname</td>
<td>A string collation, comparison, and hashing facet</td>
</tr>
<tr>
<td>compare</td>
<td>A binary function or a function object that returns true or false</td>
</tr>
<tr>
<td>complex</td>
<td>C++ complex number library</td>
</tr>
<tr>
<td>copy</td>
<td>Copies a range of elements</td>
</tr>
<tr>
<td>copy_backward</td>
<td>Copies a range of elements</td>
</tr>
<tr>
<td>count</td>
<td>Count the number of elements in a container that satisfy a given</td>
</tr>
<tr>
<td></td>
<td>condition</td>
</tr>
<tr>
<td>count_if</td>
<td>Count the number of elements in a container that satisfy a given</td>
</tr>
<tr>
<td></td>
<td>condition</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>cout</code></td>
<td>Controls output to a stream buffer associated with the object <code>stdout</code> declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td><code>ctype</code></td>
<td>A facet that includes character classification facilities</td>
</tr>
<tr>
<td><code>ctype_byname</code></td>
<td>A facet that includes character classification facilities based on the named locales</td>
</tr>
<tr>
<td><code>deque</code></td>
<td>A sequence that supports random access iterators and efficient insertion/deletion at both beginning and end</td>
</tr>
<tr>
<td><code>distance</code></td>
<td>Computes the distance between two iterators</td>
</tr>
<tr>
<td><code>divides</code></td>
<td>Returns the result of dividing its first argument by its second</td>
</tr>
<tr>
<td><code>equal</code></td>
<td>Compares two ranges for equality</td>
</tr>
<tr>
<td><code>equal_range</code></td>
<td>Finds the largest subrange in a collection into which a given value can be inserted without violating the ordering of the collection</td>
</tr>
<tr>
<td><code>equal_to</code></td>
<td>A binary function object that returns true if its first argument equals its second</td>
</tr>
<tr>
<td><code>exception</code></td>
<td>A class that supports logic and runtime errors</td>
</tr>
<tr>
<td><code>facets</code></td>
<td>A family of classes used to encapsulate categories of locale functionality</td>
</tr>
<tr>
<td><code>filebuf</code></td>
<td>Class that associates the input or output sequence with a file</td>
</tr>
<tr>
<td><code>fill</code></td>
<td>Initializes a range with a given value</td>
</tr>
<tr>
<td><code>fill_n</code></td>
<td>Initializes a range with a given value</td>
</tr>
<tr>
<td><code>find</code></td>
<td>Finds an occurrence of value in a sequence</td>
</tr>
<tr>
<td><code>find_end</code></td>
<td>Finds the last occurrence of a sub-sequence in a sequence</td>
</tr>
<tr>
<td><code>find_first_of</code></td>
<td>Finds the first occurrence of any value from one sequence in another sequence</td>
</tr>
<tr>
<td><code>find_if</code></td>
<td>Finds an occurrence of a value in a sequence that satisfies a specified predicate</td>
</tr>
<tr>
<td><code>for_each</code></td>
<td>Applies a function to each element in a range</td>
</tr>
<tr>
<td><code>fpos</code></td>
<td>Maintains position information for the iostream classes</td>
</tr>
<tr>
<td><code>front_insert_iterator</code></td>
<td>An insert iterator used to insert items at the beginning of a collection</td>
</tr>
<tr>
<td><code>front_inserter</code></td>
<td>An insert iterator used to insert items at the beginning of a collection</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fstream</td>
<td>Supports reading and writing of named files or devices associated with a file descriptor</td>
</tr>
<tr>
<td>generate</td>
<td>Initialize a container with values produced by a value-generator class</td>
</tr>
<tr>
<td>generate_n</td>
<td>Initialize a container with values produced by a value-generator class</td>
</tr>
<tr>
<td>get_temporary_buffer</td>
<td>Pointer based primitive for handling memory</td>
</tr>
<tr>
<td>greater</td>
<td>A binary function object that returns true if its first argument is greater than its second</td>
</tr>
<tr>
<td>greater_equal</td>
<td>A binary function object that returns true if its first argument is greater than or equal to its second</td>
</tr>
<tr>
<td>gslice</td>
<td>A numeric array class used to represent a generalized slice from an array</td>
</tr>
<tr>
<td>gslice_array</td>
<td>A numeric array class used to represent a BLAS-like slice from a valarray</td>
</tr>
<tr>
<td>has_facet</td>
<td>A function template used to determine if a locale has a given facet</td>
</tr>
<tr>
<td>ifstream</td>
<td>Supports reading from named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td>includes</td>
<td>A basic set of operation for sorted sequences</td>
</tr>
<tr>
<td>indirect_array</td>
<td>A numeric array class used to represent elements selected from a valarray</td>
</tr>
<tr>
<td>inner_product</td>
<td>Computes the inner product $A \times B$ of two ranges $A$ and $B$</td>
</tr>
<tr>
<td>inplace_merge</td>
<td>Merges two sorted sequences into one</td>
</tr>
<tr>
<td>insert_iterator</td>
<td>An insert iterator used to insert items into a collection rather than overwrite the collection</td>
</tr>
<tr>
<td>inserter</td>
<td>An insert iterator used to insert items into a collection rather than overwrite the collection</td>
</tr>
<tr>
<td>ios</td>
<td>A base class that includes the common functions required by all streams</td>
</tr>
<tr>
<td>ios_base</td>
<td>Defines member types and maintains data for classes that inherit from it</td>
</tr>
<tr>
<td>iosfwd</td>
<td>Declares the input/output library template classes and specializes them for wide and tiny characters</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>isalnum</td>
<td>Determines if a character is alphabetic or numeric</td>
</tr>
<tr>
<td>isalpha</td>
<td>Determines if a character is alphabetic</td>
</tr>
<tr>
<td>iscntrl</td>
<td>Determines if a character is a control character</td>
</tr>
<tr>
<td>isdigit</td>
<td>Determines if a character is a decimal digit</td>
</tr>
<tr>
<td>isgraph</td>
<td>Determines if a character is a graphic character</td>
</tr>
<tr>
<td>islower</td>
<td>Determines whether a character is lower case</td>
</tr>
<tr>
<td>isprint</td>
<td>Determines if a character is printable</td>
</tr>
<tr>
<td>ispunct</td>
<td>Determines if a character is punctuation</td>
</tr>
<tr>
<td>isspace</td>
<td>Determines if a character is a space</td>
</tr>
<tr>
<td>istream</td>
<td>Assists in reading and interpreting input from sequences controlled by a stream buffer</td>
</tr>
<tr>
<td>istream_iterator</td>
<td>A stream iterator that has iterator capabilities for istreams</td>
</tr>
<tr>
<td>istreambuf_iterator</td>
<td>Reads successive characters from the stream buffer for which it was constructed</td>
</tr>
<tr>
<td>istringstream</td>
<td>Supports reading objects of class basic_string&lt;charT,traits,Allocator&gt; from an array in memory</td>
</tr>
<tr>
<td>istringstream</td>
<td>Reads characters from an array in memory</td>
</tr>
<tr>
<td>isupper</td>
<td>Determines whether a character is upper case</td>
</tr>
<tr>
<td>isxdigit</td>
<td>Determines whether a character is a hexadecimal digit</td>
</tr>
<tr>
<td>iter_swap</td>
<td>Exchanges values in two locations</td>
</tr>
<tr>
<td>iterator</td>
<td>A base iterator class</td>
</tr>
<tr>
<td>iterator_traits</td>
<td>Returns basic information about an iterator</td>
</tr>
<tr>
<td>less</td>
<td>A binary function object that returns true if its first argument is less than its second</td>
</tr>
<tr>
<td>less_equal</td>
<td>A binary function object that returns true if its first argument is less than or equal to its second</td>
</tr>
<tr>
<td>lexicographical_compare</td>
<td>Compares two ranges lexicographically</td>
</tr>
<tr>
<td>limits</td>
<td>Refer to numeric_limits</td>
</tr>
<tr>
<td>list</td>
<td>A sequence that supports bidirectional iterators</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>locale</td>
<td>A localization class containing a polymorphic set of facets</td>
</tr>
<tr>
<td>logical_and</td>
<td>A binary function object that returns true if both of its arguments are true</td>
</tr>
<tr>
<td>logical_not</td>
<td>A unary function object that returns true if its argument is false</td>
</tr>
<tr>
<td>logical_or</td>
<td>A binary function object that returns true if either of its arguments are true</td>
</tr>
<tr>
<td>lower_bound</td>
<td>Determines the first valid position for an element in a sorted container</td>
</tr>
<tr>
<td>make_heap</td>
<td>Creates a heap</td>
</tr>
<tr>
<td>map</td>
<td>An associative container with access to non-key values using unique keys</td>
</tr>
<tr>
<td>mask_array</td>
<td>A numeric array class that gives a masked view of a valarray</td>
</tr>
<tr>
<td>max</td>
<td>Finds and returns the maximum of a pair of values</td>
</tr>
<tr>
<td>max_element</td>
<td>Finds the maximum value in a range</td>
</tr>
<tr>
<td>mem_fun</td>
<td>Function objects that adapt a pointer to a member function, to take the place of a global function</td>
</tr>
<tr>
<td>mem_fun1</td>
<td>Function objects that adapt a pointer to a member function, to take the place of a global function</td>
</tr>
<tr>
<td>mem_fun_ref</td>
<td>Function objects that adapt a pointer to a member function, to take the place of a global function</td>
</tr>
<tr>
<td>mem_fun_ref1</td>
<td>Function objects that adapt a pointer to a member function, to take the place of a global function</td>
</tr>
<tr>
<td>merge</td>
<td>Merges two sorted sequences into a third sequence</td>
</tr>
<tr>
<td>messages</td>
<td>Messaging facets</td>
</tr>
<tr>
<td>messages_byname</td>
<td>Messaging facets</td>
</tr>
<tr>
<td>min</td>
<td>Finds and returns the minimum of a pair of values</td>
</tr>
<tr>
<td>min_element</td>
<td>Finds the minimum value in a range</td>
</tr>
<tr>
<td>minus</td>
<td>Returns the result of subtracting its second argument from its first</td>
</tr>
<tr>
<td>mismatch</td>
<td>Compares elements from two sequences and returns the first two elements that don’t match each other</td>
</tr>
<tr>
<td>modulus</td>
<td>Returns the remainder obtained by dividing the first argument by the second argument</td>
</tr>
</tbody>
</table>
### TABLE 13-2 Man Pages for C++ Standard Library (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>money_get</td>
<td>Monetary formatting facet for input</td>
</tr>
<tr>
<td>money_put</td>
<td>Monetary formatting facet for output</td>
</tr>
<tr>
<td>moneypunct</td>
<td>Monetary punctuation facets</td>
</tr>
<tr>
<td>moneypunctbyname</td>
<td>Monetary punctuation facets</td>
</tr>
<tr>
<td>multimap</td>
<td>An associative container that gives access to non-key values using keys</td>
</tr>
<tr>
<td>multiplies</td>
<td>A binary function object that returns the result of multiplying its first and second arguments</td>
</tr>
<tr>
<td>multiset</td>
<td>An associative container that allows fast access to stored key values</td>
</tr>
<tr>
<td>negate</td>
<td>Unary function object that returns the negation of its argument</td>
</tr>
<tr>
<td>next_permutation</td>
<td>Generates successive permutations of a sequence based on an ordering function</td>
</tr>
<tr>
<td>not1</td>
<td>A function adaptor used to reverse the sense of a unary predicate function object</td>
</tr>
<tr>
<td>not2</td>
<td>A function adaptor used to reverse the sense of a binary predicate function object</td>
</tr>
<tr>
<td>not_equal_to</td>
<td>A binary function object that returns true if its first argument is not equal to its second</td>
</tr>
<tr>
<td>nth_element</td>
<td>Rearranges a collection so that all elements lower in sorted order than the nth element come before it and all elements higher in sorted order than the nth element come after it</td>
</tr>
<tr>
<td>num_get</td>
<td>A numeric formatting facet for input</td>
</tr>
<tr>
<td>num_put</td>
<td>A numeric formatting facet for output</td>
</tr>
<tr>
<td>numeric_limits</td>
<td>A class for representing information about scalar types</td>
</tr>
<tr>
<td>numpunct</td>
<td>A numeric punctuation facet</td>
</tr>
<tr>
<td>numpunctbyname</td>
<td>A numeric punctuation facet</td>
</tr>
<tr>
<td>ofstream</td>
<td>Supports writing into named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td>ostream</td>
<td>Assists in formatting and writing output to sequences controlled by a stream buffer</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>ostream_iterator</code></td>
<td>Stream iterators allow for use of iterators with ostrams and istreams</td>
</tr>
<tr>
<td><code>ostreambuf_iterator</code></td>
<td>Writes successive characters onto the stream buffer object from which it was constructed</td>
</tr>
<tr>
<td><code>ostringstream</code></td>
<td>Supports writing objects of class <code>basic_string&lt;charT,traits,Allocator&gt;</code></td>
</tr>
<tr>
<td><code>ostrstream</code></td>
<td>Writes to an array in memory</td>
</tr>
<tr>
<td><code>pair</code></td>
<td>A template for heterogeneous pairs of values</td>
</tr>
<tr>
<td><code>partial_sort</code></td>
<td>Templatized algorithm for sorting collections of entities</td>
</tr>
<tr>
<td><code>partial_sort_copy</code></td>
<td>Templatized algorithm for sorting collections of entities</td>
</tr>
<tr>
<td><code>partial_sum</code></td>
<td>Calculates successive partial sums of a range of values</td>
</tr>
<tr>
<td><code>partition</code></td>
<td>Places all of the entities that satisfy the given predicate before all of the entities that do not</td>
</tr>
<tr>
<td><code>permutation</code></td>
<td>Generates successive permutations of a sequence based on an ordering function</td>
</tr>
<tr>
<td><code>plus</code></td>
<td>A binary function object that returns the result of adding its first and second arguments</td>
</tr>
<tr>
<td><code>pointer_to_binary_function</code></td>
<td>A function object that adapts a pointer to a binary function, to take the place of a <code>binary_function</code></td>
</tr>
<tr>
<td><code>pointer_to_unary_function</code></td>
<td>A function object class that adapts a pointer to a function, to take the place of a <code>unary_function</code></td>
</tr>
<tr>
<td><code>pop_heap</code></td>
<td>Moves the largest element off the heap</td>
</tr>
<tr>
<td><code>prev_permutation</code></td>
<td>Generates successive permutations of a sequence based on an ordering function</td>
</tr>
<tr>
<td><code>priority_queue</code></td>
<td>A container adapter that behaves like a priority queue</td>
</tr>
<tr>
<td><code>ptr_fun</code></td>
<td>A function that is overloaded to adapt a pointer to a function, to take the place of a function</td>
</tr>
<tr>
<td><code>push_heap</code></td>
<td>Places a new element into a heap</td>
</tr>
<tr>
<td><code>queue</code></td>
<td>A container adaptor that behaves like a queue (first in, first out)</td>
</tr>
<tr>
<td><code>random_shuffle</code></td>
<td>Randomly shuffles elements of a collection</td>
</tr>
<tr>
<td><code>raw_storage_iterator</code></td>
<td>Enables iterator-based algorithms to store results into uninitialized memory</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>remove</td>
<td>Moves desired elements to the front of a container, and returns an iterator that describes where the sequence of desired elements ends</td>
</tr>
<tr>
<td>remove_copy</td>
<td>Moves desired elements to the front of a container, and returns an iterator that describes where the sequence of desired elements ends</td>
</tr>
<tr>
<td>remove_copy_if</td>
<td>Moves desired elements to the front of a container, and returns an iterator that describes where the sequence of desired elements ends</td>
</tr>
<tr>
<td>remove_if</td>
<td>Moves desired elements to the front of a container, and returns an iterator that describes where the sequence of desired elements ends</td>
</tr>
<tr>
<td>replace</td>
<td>Substitutes elements in a collection with new values</td>
</tr>
<tr>
<td>replace_copy</td>
<td>Substitutes elements in a collection with new values, and moves the revised sequence into result</td>
</tr>
<tr>
<td>replace_copy_if</td>
<td>Substitutes elements in a collection with new values, and moves the revised sequence into result</td>
</tr>
<tr>
<td>replace_if</td>
<td>Substitutes elements in a collection with new values</td>
</tr>
<tr>
<td>return Temporary_buffer</td>
<td>A pointer-based primitive for handling memory</td>
</tr>
<tr>
<td>reverse</td>
<td>Reverses the order of elements in a collection</td>
</tr>
<tr>
<td>reverse_copy</td>
<td>Reverses the order of elements in a collection while copying them to a new collection</td>
</tr>
<tr>
<td>reverse_iterator</td>
<td>An iterator that traverses a collection backwards</td>
</tr>
<tr>
<td>rotate</td>
<td>Swaps the segment that contains elements from first through middle-minus-one with the segment that contains the elements from middle through last</td>
</tr>
<tr>
<td>rotate_copy</td>
<td>Swaps the segment that contains elements from first through middle-minus-one with the segment that contains the elements from middle through last</td>
</tr>
<tr>
<td>search</td>
<td>Finds a sub-sequence within a sequence of values that is element-wise equal to the values in an indicated range</td>
</tr>
<tr>
<td>search_n</td>
<td>Finds a sub-sequence within a sequence of values that is element-wise equal to the values in an indicated range</td>
</tr>
<tr>
<td>set</td>
<td>An associative container that supports unique keys</td>
</tr>
<tr>
<td>set_difference</td>
<td>A basic set operation for constructing a sorted difference</td>
</tr>
</tbody>
</table>
### TABLE 13–2 Man Pages for C++ Standard Library (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_intersection</td>
<td>A basic set operation for constructing a sorted intersection</td>
</tr>
<tr>
<td>set_symmetric_difference</td>
<td>A basic set operation for constructing a sorted symmetric difference</td>
</tr>
<tr>
<td>set_union</td>
<td>A basic set operation for constructing a sorted union</td>
</tr>
<tr>
<td>slice</td>
<td>A numeric array class for representing a BLAS-like slice from an array</td>
</tr>
<tr>
<td>slice_array</td>
<td>A numeric array class for representing a BLAS-like slice from a valarray</td>
</tr>
<tr>
<td>smanip</td>
<td>Helper classes used to implement parameterized manipulators</td>
</tr>
<tr>
<td>smanip_fill</td>
<td>Helper classes used to implement parameterized manipulators</td>
</tr>
<tr>
<td>sort</td>
<td>A templatized algorithm for sorting collections of entities</td>
</tr>
<tr>
<td>sort_heap</td>
<td>Converts a heap into a sorted collection</td>
</tr>
<tr>
<td>stable_partition</td>
<td>Places all of the entities that satisfy the given predicate before all of the entities that do not, while maintaining the relative order of elements in each group</td>
</tr>
<tr>
<td>stable_sort</td>
<td>A templatized algorithm for sorting collections of entities</td>
</tr>
<tr>
<td>stack</td>
<td>A container adapter that behaves like a stack (last in, first out)</td>
</tr>
<tr>
<td>streambuf</td>
<td>Abstract base class for deriving various stream buffers to facilitate control of character sequences</td>
</tr>
<tr>
<td>string</td>
<td>A typedef for basic_string&lt;char, char_traits&lt;char&gt;, allocator&lt;char&gt;&gt;</td>
</tr>
<tr>
<td>stringbuf</td>
<td>Associates the input or output sequence with a sequence of arbitrary characters</td>
</tr>
<tr>
<td>stringstream</td>
<td>Supports writing and reading objects of class basic_string&lt;charT, traits, Allocator&gt; to/from an array in memory</td>
</tr>
<tr>
<td>strstream</td>
<td>Reads and writes to an array in memory</td>
</tr>
<tr>
<td>strstreambuf</td>
<td>Associates either the input sequence or the output sequence with a tiny character array whose elements store arbitrary values</td>
</tr>
<tr>
<td>swap</td>
<td>Exchanges values</td>
</tr>
<tr>
<td>swap_ranges</td>
<td>Exchanges a range of values in one location with those in another</td>
</tr>
<tr>
<td>time_get</td>
<td>A time formatting facet for input</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>time_getbyname</td>
<td>A time formatting facet for input, based on the named locales</td>
</tr>
<tr>
<td>time_put</td>
<td>A time formatting facet for output</td>
</tr>
<tr>
<td>time_putbyname</td>
<td>A time formatting facet for output, based on the named locales</td>
</tr>
<tr>
<td>tolower</td>
<td>Converts a character to lower case</td>
</tr>
<tr>
<td>toupper</td>
<td>Converts a character to upper case</td>
</tr>
<tr>
<td>transform</td>
<td>Applies an operation to a range of values in a collection and stores the result</td>
</tr>
<tr>
<td>unary_function</td>
<td>A base class for creating unary function objects</td>
</tr>
<tr>
<td>unary_negate</td>
<td>A function object that returns the complement of the result of its unary predicate</td>
</tr>
<tr>
<td>uninitialized_copy</td>
<td>An algorithm that uses construct to copy values from one range to another location</td>
</tr>
<tr>
<td>uninitialized_fill</td>
<td>An algorithm that uses the construct algorithm for setting values in a collection</td>
</tr>
<tr>
<td>uninitialized_fill_n</td>
<td>An algorithm that uses the construct algorithm for setting values in a collection</td>
</tr>
<tr>
<td>unique</td>
<td>Removes consecutive duplicates from a range of values and places the resulting unique values into the result</td>
</tr>
<tr>
<td>unique_copy</td>
<td>Removes consecutive duplicates from a range of values and places the resulting unique values into the result</td>
</tr>
<tr>
<td>upper_bound</td>
<td>Determines the last valid position for a value in a sorted container</td>
</tr>
<tr>
<td>use_facet</td>
<td>A template function used to obtain a facet</td>
</tr>
<tr>
<td>valarray</td>
<td>An optimized array class for numeric operations</td>
</tr>
<tr>
<td>vector</td>
<td>A sequence that supports random access iterators</td>
</tr>
<tr>
<td>wcerr</td>
<td>Controls output to an unbuffered stream buffer associated with the object stderr declared in &lt;cstdio&gt;</td>
</tr>
<tr>
<td>wcin</td>
<td>Controls input from a stream buffer associated with the object stdin declared in &lt;cstdio&gt;</td>
</tr>
<tr>
<td>wclog</td>
<td>Controls output to a stream buffer associated with the object stderr declared in &lt;cstdio&gt;</td>
</tr>
<tr>
<td>wcput</td>
<td>Controls output to a stream buffer associated with the object stdout declared in &lt;cstdio&gt;</td>
</tr>
</tbody>
</table>
### Man Pages for C++ Standard Library (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wfilebuf</code></td>
<td>Class that associates the input or output sequence with a file</td>
</tr>
<tr>
<td><code>wfstream</code></td>
<td>Supports reading and writing of named files or devices associated with a file descriptor</td>
</tr>
<tr>
<td><code>wifstream</code></td>
<td>Supports reading from named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td><code>wios</code></td>
<td>A base class that includes the common functions required by all streams</td>
</tr>
<tr>
<td><code>wiostream</code></td>
<td>Assists in reading and interpreting input from sequences controlled by a stream buffer</td>
</tr>
<tr>
<td><code>wistringstream</code></td>
<td>Supports reading objects of class <code>basic_string&lt;charT, traits, Allocator&gt;</code> from an array in memory</td>
</tr>
<tr>
<td><code>wofstream</code></td>
<td>Supports writing into named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td><code>wostream</code></td>
<td>Assists in formatting and writing output to sequences controlled by a stream buffer</td>
</tr>
<tr>
<td><code>wstringstream</code></td>
<td>Supports writing objects of class <code>basic_string&lt;charT, traits, Allocator&gt;</code></td>
</tr>
<tr>
<td><code>wstreambuf</code></td>
<td>Abstract base class for deriving various stream buffers to facilitate control of character sequences</td>
</tr>
<tr>
<td><code>wstring</code></td>
<td>A typedef for <code>basic_string&lt;wchar_t, char_traits&lt;wchar_t&gt;, allocator&lt;wchar_t&gt;&gt;</code></td>
</tr>
<tr>
<td><code>wstrstream</code></td>
<td>Associates the input or output sequence with a sequence of arbitrary characters</td>
</tr>
</tbody>
</table>

### 13.3 STLport

Use the STLport implementation of the standard library if you wish to use an alternative standard library to `libCstd`. You can issue the following compiler option to turn off `libCstd` and use the STLport library instead:

- `-library=stlport4`

See "A.2.50 -library=[,]..." on page 252 for more information.

This release includes both a static archive called `libstlport.a` and a dynamic library called `libstlport.so`. 
Consider the following information before you decide whether or not you are going to use the STLport implementation:

- STLport is an open source product and does not guarantee compatibility across different releases. In other words, compiling with a future version of STLport may break applications compiled with STLport 4.5.3. It also might not be possible to link binaries compiled using STLport 4.5.3 with binaries compiled using a future version of STLport.

- The stlport4, Cstd and iostream libraries provide their own implementation of I/O streams. Specifying more than one of these with the -library option can result in undefined program behavior.

- Future releases of the compiler might not include STLport4. They might include only a later version of STLport. The compiler option -library=stlport4 might not be available in future releases, but could be replaced by an option referring to a later STLport version.

- Tools.h++ is not supported with STLport.

- STLport is binary incompatible with the default libCstd. If you use the STLport implementation of the standard library, then you must compile and link all files, including third-party libraries, with the option -library=stlport4. This means, for example, that you cannot use the STLport implementation and the C++ interval math library libCsunimath together. The reason for this is that libCsunimath was compiled with the default library headers, not with STLport.

- If you decide to use the STLport implementation, be certain to include header files that your code implicitly references. The standard headers are allowed, but not required, to include one another as part of the implementation.

- You cannot use the STLport implementation if you compile with -compat=4.

### 13.3.1 Redistribution and Supported STLport Libraries

See the Runtime Libraries Readme for a list of libraries and object files that you can redistribute with your executables or libraries under the terms of the End User Object Code License. The C++ section of this readme lists which version of the STLport.so this release of the compiler supports. This readme is available as part of the product documentation on the Sun Studio SDN portal at [http://developers.sun.com/sunstudio/documentation/](http://developers.sun.com/sunstudio/documentation/)

The following test case does not compile with STLport because the code in the test case makes unportable assumptions about the library implementation. In particular, it assumes that either <vector> or <iostream> automatically include <iterator>, which is not a valid assumption.

```cpp
#include <vector>
#include <iostream>

using namespace std;
```
int main ()
{
    vector <int> v1 (10);
    vector <int> v3 (v1.size());
    for (int i = 0; i < v1.size (); i++)
    {v1[i] = i; v3[i] = i;}
    vector <int> v2(v1.size ());
    copy_backward (v1.begin (), v1.end (), v2.end ());
    ostream_iterator<int> iter (cout, " ");
    copy (v2.begin (), v2.end (), iter);
    cout << endl;
    return 0;
}

To fix the problem, include <iterator> in the source.
C++, like C, has no built-in input or output statements. Instead, I/O facilities are provided by a library. The C++ compiler provides both the classic implementation and the ISO standard implementation of the `iostream` classes.

- In compatibility mode (`-compat=4`), the classic `iostream` classes are contained in `libC`.
- In standard mode (default mode), the classic `iostream` classes are contained in `libiostream`. Use `libiostream` when you have source code that uses the classic `iostream` classes and you want to compile the source in standard mode. To use the classic `iostream` facilities in standard mode, include the `iostream.h` header file and compile using the `-library=iostream` option.
- The standard `iostream` classes are available only in standard mode, and are contained in the C++ standard library, `libCstd`.

This chapter provides an introduction to the classic `iostream` library and provides examples of its use. This chapter does not provide a complete description of the `iostream` library. See the `iostream` library man pages for more details. To access the classic `iostream` man pages type: `man -s 3CC4 name`

### 14.1 Predefined `iostreams`

There are four predefined `iostreams`:
- `cin`, connected to standard input
- `cout`, connected to standard output
- `cerr`, connected to standard error
- `clog`, connected to standard error

The predefined `iostreams` are fully buffered, except for `cerr`. See “14.3.1 Output Using `iostream`” on page 173 and “14.3.2 Input Using `iostream`” on page 176.
14.2 Basic Structure of iostream Interaction

By including the iostream library, a program can use any number of input or output streams. Each stream has some source or sink, which may be one of the following:

- Standard input
- Standard output
- Standard error
- A file
- An array of characters

A stream can be restricted to input or output, or a single stream can allow both input and output. The iostream library implements these streams using two processing layers.

- The lower layer implements sequences, which are simply streams of characters. These sequences are implemented by the streambuf class, or by classes derived from it.
- The upper layer performs formatting operations on sequences. These formatting operations are implemented by the istream and ostream classes, which have as a member an object of a type derived from class streambuf. An additional class, iostream, is for streams on which both input and output can be performed.

Standard input, output, and error are handled by special class objects derived from class istream or ostream.

The ifstream, ofstream, and fstream classes, which are derived from istream, ostream, and iostream respectively, handle input and output with files.

The istrstream, ostrstream, and strstream classes, which are derived from istream, ostream, and iostream respectively, handle input and output to and from arrays of characters.

When you open an input or output stream, you create an object of one of these types, and associate the streambuf member of the stream with a device or file. You generally do this association through the stream constructor, so you don’t work with the streambuf directly. The iostream library predefines stream objects for the standard input, standard output, and error output, so you don’t have to create your own objects for those streams.

You use operators or iostream member functions to insert data into a stream (output) or extract data from a stream (input), and to control the format of data that you insert or extract.

When you want to insert and extract a new data type—one of your classes—you generally overload the insertion and extraction operators.
14.3 Using the Classic `iostream` Library

To use routines from the classic `iostream` library, you must include the header files for the part of the library you need. The header files are described in the following table.

<table>
<thead>
<tr>
<th>Header File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iostream.h</code></td>
<td>Declares basic features of <code>iostream</code> library.</td>
</tr>
<tr>
<td><code>fstream.h</code></td>
<td>Declares <code>iostreams</code> and <code>streambufs</code> specialized to files. Includes <code>iostream.h</code>.</td>
</tr>
<tr>
<td><code>strstream.h</code></td>
<td>Declares <code>iostreams</code> and <code>streambufs</code> specialized to character arrays. Includes <code>iostream.h</code>.</td>
</tr>
<tr>
<td><code>iomanip.h</code></td>
<td>Declares manipulators: values you insert into or extract from <code>iostreams</code> to have different effects. Includes <code>iostream.h</code>.</td>
</tr>
<tr>
<td><code>stdiostream.h</code></td>
<td>(obsolete) Declares <code>iostreams</code> and <code>streambufs</code> specialized to use <code>stdio</code> <code>FILEs</code>. Includes <code>iostream.h</code>.</td>
</tr>
<tr>
<td><code>stream.h</code></td>
<td>(obsolete) Includes <code>iostream.h</code>, <code>fstream.h</code>, <code>iomanip.h</code>, and <code>stdiostream.h</code>. For compatibility with old-style streams from C++ version 1.2.</td>
</tr>
</tbody>
</table>

You usually do not need all of these header files in your program. Include only the ones that contain the declarations you need. In compatibility mode (`-compat=[4]`), the classic `iostream` library is part of `libC`, and is linked automatically by the `CC` driver. In standard mode (the default), `libiostream` contains the classic `iostream` library.

14.3.1 Output Using `iostream`

Output using `iostream` usually relies on the overloaded left-shift operator (`<<`) which, in the context of `iostream`, is called the insertion operator. To output a value to standard output, you insert the value in the predefined output stream `cout`. For example, given a value `someValue`, you send it to standard output with a statement like:

```cpp
cout << someValue;
```

The insertion operator is overloaded for all built-in types, and the value represented by `someValue` is converted to its proper output representation. If, for example, `someValue` is a `float` value, the `<<` operator converts the value to the proper sequence of digits with a decimal point. Where it inserts `float` values on the output stream, `<<` is called the float inserter. In general, given a type `X`, `<<` is called the `X` inserter. The format of output and how you can control it is discussed in the `ios(3CC4)` man page.
The `iostream` library does not support user-defined types. If you define types that you want to output in your own way, you must define an inserter (that is, overload the `<<` operator) to handle them correctly.

The `<<` operator can be applied repetitively. To insert two values on `cout`, you can use a statement like the one in the following example:

```cpp
cout << someValue << anotherValue;
```

The output from the above example will show no space between the two values. So you may want to write the code this way:

```cpp
cout << someValue << " " << anotherValue;
```

The `<<` operator has the precedence of the left shift operator (its built-in meaning). As with other operators, you can always use parentheses to specify the order of action. It is often a good idea to use parentheses to avoid problems of precedence. Of the following four statements, the first two are equivalent, but the last two are not.

```cpp
cout << a+b;       // + has higher precedence than <<
cout << (a+b);
cout << (a&y);    // << has precedence higher than &
cout << a&y;      // probably an error: (cout << a) & y
```

### 14.3.1.1 Defining Your Own Insertion Operator

The following example defines a `string` class:

```cpp
#include <stdlib.h>
#include <iostream.h>

class string {
private:
    char* data;
    size_t size;

public:
    // (functions not relevant here)
    friend ostream& operator<<(ostream&, const string&);
    friend istream& operator>>(istream&, string&);
};
```

The insertion and extraction operators must in this case be defined as friends because the data part of the `string` class is private.
ostream& operator<<(ostream& ostr, const string& output)
{
    return ostr << output.data;
}

Here is the definition of operator<< overloaded for use with strings.

cout << string1 << string2;

operator<< takes ostream& (that is, a reference to an ostream) as its first argument and returns
the same ostream, making it possible to combine insertions in one statement.

14.3.1.2 Handling Output Errors

Generally, you don’t have to check for errors when you overload operator<< because the
iostream library is arranged to propagate errors.

When an error occurs, the iostream where it occurred enters an error state. Bits in the
iostream’s state are set according to the general category of the error. The inserters defined in
iostream ignore attempts to insert data into any stream that is in an error state, so such
attempts do not change the iostream’s state.

In general, the recommended way to handle errors is to periodically check the state of the
output stream in some central place. If there is an error, you should handle it in some way. This
chapter assumes that you define a function error, which takes a string and aborts the program.
error is not a predefined function. See “14.3.9 Handling Input Errors” on page 179 for an
example of an error function. You can examine the state of an iostream with the operator
!, which returns a nonzero value if the iostream is in an error state. For example:

if (!cout) error("output error");

There is another way to test for errors. The ios class defines operator void *(), so it returns a
NULL pointer when there is an error. You can use a statement like:

if (cout << x) return; // return if successful

You can also use the function good, a member of ios:

if (cout.good()) return; // return if successful

The error bits are declared in the enum:

enum io_state {goodbit=0, eofbit=1, failbit=2,
badbit=4, hardfail=0x80};

For details on the error functions, see the iostream man pages.
14.3.3 Flushing

As with most I/O libraries, `iostream` often accumulates output and sends it on in larger and generally more efficient chunks. If you want to flush the buffer, you simply insert the special value `flush`. For example:

```cpp
cout << "This needs to get out immediately." << flush;
```

`flush` is an example of a kind of object known as a manipulator, which is a value that can be inserted into a `iostream` to have some effect other than causing output of its value. It is really a function that takes an `ostream&` or `istream&` argument and returns its argument after performing some actions on it (see “14.7 Manipulators” on page 183).

14.3.4 Binary Output

To obtain output in the raw binary form of a value, use the member function `write` as shown in the following example. This example shows the output in the raw binary form of `x`.

```cpp
cout.write((char*)&x, sizeof(x));
```

The previous example violates type discipline by converting `&x` to `char*`. Doing so is normally harmless, but if the type of `x` is a class with pointers, virtual member functions, or one that requires nontrivial constructor actions, the value written by the above example cannot be read back in properly.

14.3.2 Input Using `iostream`

Input using `iostream` is similar to output. You use the extraction operator `>>` and you can string together extractions the way you can with insertions. For example:

```cpp
cin >> a >> b;
```

This statement gets two values from standard input. As with other overloaded operators, the extractors used depend on the types of `a` and `b` (and two different extractors are used if `a` and `b` have different types). The format of input and how you can control it is discussed in some detail in the `ios(3CC4)` man page. In general, leading whitespace characters (spaces, newlines, tabs, form-feeds, and so on) are ignored.

14.3.3 Defining Your Own Extraction Operators

When you want input for a new type, you overload the extraction operator for it, just as you overload the insertion operator for output.

Class `string` defines its extraction operator in the following code example:
EXAMPLE 14–1  string Extraction Operator

```cpp
istream& operator>>(istream& istr, string& input) {
    const int maxline = 256;
    char holder[maxline];
    istr.get(holder, maxline, "\n");
    input = holder;
    return istr;
}
```

The `get` function reads characters from the input stream `istr` and stores them in `holder` until `maxline-1` characters have been read, or a new line is encountered, or EOF, whichever happens first. The data in `holder` is then null-terminated. Finally, the characters in `holder` are copied into the target string.

By convention, an extractor converts characters from its first argument (in this case, `istream& istr`), stores them in its second argument, which is always a reference, and returns its first argument. The second argument must be a reference because an extractor is meant to store the input value in its second argument.

### 14.3.4 Using the char* Extractor

This predefined extractor is mentioned here because it can cause problems. Use it like this:

```cpp
char x[50];
cin >> x;
```

This extractor skips leading whitespace and extracts characters and copies them to `x` until it reaches another whitespace character. It then completes the string with a terminating null (0) character. Be careful, because input can overflow the given array.

You must also be sure the pointer points to allocated storage. For example, here is a common error:

```cpp
char * p; // not initialized
cin >> p;
```

There is no telling where the input data will be stored, and it may cause your program to abort.

### 14.3.5 Reading Any Single Character

In addition to using the char extractor, you can get a single character with either form of the `get` member function. For example:
14.3.6 Binary Input
If you need to read binary values (such as those written with the member function write), you can use the read member function. The following example shows how to input the raw binary form of x using the read member function, and is the inverse of the earlier example that uses write.

```cpp
cin.read((char*)&x, sizeof(x));
```

14.3.7 Peeking at Input
You can use the peek member function to look at the next character in the stream without extracting it. For example:

```cpp
if (cin.peek()!= c) return 0;
```

14.3.8 Extracting Whitespace
By default, the iostream extractors skip leading whitespace. You can turn off the skip flag to prevent this from happening. The following example turns off whitespace skipping from cin, then turns it back on:

```cpp
cin.unsetf(ios::skipws); // turn off whitespace skipping
...
// turn it on again
```
You can use the iostream manipulator ws to remove leading whitespace from the iostream, whether or not skipping is enabled. The following example shows how to remove the leading whitespace from iostream istr:

istr >> ws;

### 14.3.9 Handling Input Errors

By convention, an extractor whose first argument has a nonzero error state should not extract anything from the input stream and should not clear any error bits. An extractor that fails should set at least one error bit.

As with output errors, you should check the error state periodically and take some action, such as aborting, when you find a nonzero state. The ! operator tests the error state of an iostream. For example, the following code produces an input error if you type alphabetic characters for input:

```c++
#include <stdlib.h>
#include <iostream.h>

void error (const char* message) {
    cerr << message << "\n";
    exit(1);
}

int main() {
    cout << "Enter some characters: ";
    int bad;
    cin >> bad;
    if (!cin) error("aborted due to input error");
    cout << "If you see this, not an error.\n";
    return 0;
}
```

Class ios has member functions that you can use for error handling. See the man pages for details.

### 14.3.10 Using iostreams With stdio

You can use stdio with C++ programs, but problems can occur when you mix iostreams and stdio in the same standard stream within a program. For example, if you write to both stdout and cout, independent buffering occurs and produces unexpected results. The problem is worse if you input from both stdin and cin, since independent buffering may turn the input into trash.
To eliminate this problem with standard input, standard output and standard error, use the following instruction before performing any input or output. It connects all the predefined iostreams with the corresponding predefined stdio FILEs.

```cpp
ios::sync_with_stdio();
```

Such a connection is not the default because there is a significant performance penalty when the predefined streams are made unbuffered as part of the connection. You can use both stdio and iostreams in the same program applied to different files. That is, you can write to stdout using stdio routines and write to other files attached to iostreams. You can open stdio FILEs for input and also read from cin so long as you don’t also try to read from stdin.

### 14.4 Creating iostreams

To read or write a stream other than the predefined iostreams, you need to create your own iostream. In general, that means creating objects of types defined in the iostream library. This section discusses the various types available.

#### 14.4.1 Dealing With Files Using Class fstream

Dealing with files is similar to dealing with standard input and standard output; classes ifstream, ofstream, and fstream are derived from classes istream, ostream, and iostream, respectively. As derived classes, they inherit the insertion and extraction operations (along with the other member functions) and also have members and constructors for use with files.

Include the file fstream.h to use any of the fstreams. Use an ifstream when you only want to perform input, an ofstream for output only, and an fstream for a stream on which you want to perform both input and output. Use the name of the file as the constructor argument.

For example, copy the file thisFile to the file thatFile as in the following example:

```cpp
ifstream fromFile("thisFile");
if (!fromFile)
    error("unable to open 'thisFile' for input");
ofstream toFile ("thatFile");
if (!toFile)
    error("unable to open 'thatFile' for output");
char c;
while (toFile & fromFile.get(c)) toFile.put(c);
```

This code:

- Creates an ifstream object called fromFile with a default mode of ios::in and connects it to thisFile. It opens thisFile.
Check the error state of the new ifstream object and, if it is in a failed state, calls the error function, which must be defined elsewhere in the program.

Creates an ofstream object called toFile with a default mode of ios::out and connects it to thatFile.

Checks the error state of toFile as above.

Creates a char variable to hold the data while it is passed.

Copies the contents of fromFile to toFile one character at a time.

Note – It is, of course, undesirable to copy a file this way, one character at a time. This code is provided just as an example of using fstreams. You should instead insert the streambuf associated with the input stream into the output stream. See “14.10 Streambufs” on page 187, and the man page sbufpub(3CC4).

### 14.4.1.1 Open Mode

The mode is constructed by or-ing bits from the enumerated type open_mode, which is a public type of class ios and has the definition:

```cpp
enum open_mode {binary=0, in=1, out=2, ate=4, app=8, trunc=0x10, nocreates=0x20, noreplace=0x40};
```

Note – The binary flag is not needed on UNIX, but is provided for compatibility with systems that do need it. Portable code should use the binary flag when opening binary files.

You can open a file for both input and output. For example, the following code opens file someName for both input and output, attaching it to the fstream variable inoutFile.

```cpp
fstreamInOutFile("someName", ios::in|ios::out);
```

### 14.4.1.2 Declaring an ofstream Without Specifying a File

You can declare an ofstream without specifying a file and open the file later. For example, the following creates the ofstream toFile for writing.

```cpp
ofstream toFile;
toFile.open(argv[1], ios::out);
```

### 14.4.1.3 Opening and Closing Files

You can close the fstream and then open it with another file. For example, to process a list of files provided on the command line:
ifstream infile;
for (char** f = &argv[1]; *f; ++f) {
    infile.open(*f, ios::in);
    ...
    infile.close();
}

14.4.1.4 Opening a File Using a File Descriptor

If you know a file descriptor, such as the integer 1 for standard output, you can open it like this:

ofstream outfile;
outfile.attach(1);

When you open a file by providing its name to one of the fstream constructors or by using the open function, the file is automatically closed when the fstream is destroyed (by a delete or when it goes out of scope). When you attach a file to an fstream, it is not automatically closed.

14.4.1.5 Repositioning Within a File

You can alter the reading and writing position in a file. Several tools are supplied for this purpose:

- streampos is a type that can record a position in an iostream.
-tellg (tellp) is an istream (ostream) member function that reports the file position. Since istream and ostream are the parent classes of fstream, tellg and tellp can also be invoked as a member function of the fstream class.
- seekg (seekp) is an istream (ostream) member function that finds a given position.
- The seek_dir enum specifies relative positions for use with seek.

enum seek_dir {beg=0, cur=1, end=2};

For example, given an fstream aFile:

streampos original = aFile.tellp();   //save current position
aFile.seekp(0, ios::end);   //reposition to end of file
aFile << x;   //write a value to file
aFile.seekp(original);   //return to original position

seekg (seekp) can take one or two parameters. When it has two parameters, the first is a position relative to the position indicated by the seek_dir value given as the second parameter. For example:

aFile.seekp(-10, ios::end);

moves to 10 bytes from the end while
aFile.seekp(10, ios::cur);

moves to 10 bytes forward from the current position.

**Note** – Arbitrary seeks on text streams are not portable, but you can always return to a previously saved streampos value.

### 14.5 Assignment of iostreams

iostreams does not allow assignment of one stream to another.

The problem with copying a stream object is that there are then two versions of the state information, such as a pointer to the current write position within an output file, which can be changed independently. As a result, problems could occur.

### 14.6 Format Control

Format control is discussed in detail in the in the man page ios(3CC4).

### 14.7 Manipulators

Manipulators are values that you can insert into or extract from iostreams to have special effects.

Parameterized manipulators are manipulators that take one or more parameters.

Because manipulators are ordinary identifiers, and therefore use up possible names, iostream doesn’t define them for every possible function. A number of manipulators are discussed with member functions in other parts of this chapter.

There are 13 predefined manipulators, as described in Table 14–2. When using that table, assume the following:

- `i` has type `long`.
- `n` has type `int`.
- `c` has type `char`.
- `istr` is an input stream.
- `ostr` is an output stream.
### TABLE 14–2 iostream Predefined Manipulators

<table>
<thead>
<tr>
<th>Predefined Manipulator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ostr &lt;&lt; dec, istr &gt;&gt; dec</td>
<td>Makes the integer conversion base 10.</td>
</tr>
<tr>
<td>2 ostr &lt;&lt; endl</td>
<td>Inserts a newline character (‘\n’) and invokes ostream::flush().</td>
</tr>
<tr>
<td>3 ostr &lt;&lt; ends</td>
<td>Inserts a null (0) character. Useful when dealing with strstream.</td>
</tr>
<tr>
<td>4 ostr &lt;&lt; flush</td>
<td>Invokes ostream::flush().</td>
</tr>
<tr>
<td>5 ostr &lt;&lt; hex, istr &gt;&gt; hex</td>
<td>Makes the integer conversion base 16.</td>
</tr>
<tr>
<td>6 ostr &lt;&lt; oct, istr &gt;&gt; oct</td>
<td>Make the integer conversion base 8.</td>
</tr>
<tr>
<td>7 istr &gt;&gt; ws</td>
<td>Extracts whitespace characters (skips whitespace) until a non-whitespace character is found (which is left in istr).</td>
</tr>
<tr>
<td>8 ostr &lt;&lt; setbase(n), istr &gt;&gt; setbase(n)</td>
<td>Sets the conversion base to n (0, 8, 10, 16 only).</td>
</tr>
<tr>
<td>9 ostr &lt;&lt; setw(n), istr &gt;&gt; setw(n)</td>
<td>Invokes ios::width(n). Sets the field width to n.</td>
</tr>
<tr>
<td>10 ostr &lt;&lt; resetiosflags(i), istr &gt;&gt; resetiosflags(i)</td>
<td>Clears the flags bitvector according to the bits set in i.</td>
</tr>
<tr>
<td>11 ostr &lt;&lt; setiosflags(i), istr &gt;&gt; setiosflags(i)</td>
<td>Sets the flags bitvector according to the bits set in i.</td>
</tr>
<tr>
<td>12 ostr &lt;&lt; setfill(c), istr &gt;&gt; setfill(c)</td>
<td>Sets the fill character (for padding a field) to c.</td>
</tr>
<tr>
<td>13 ostr &lt;&lt; setprecision(n), istr &gt;&gt; setprecision(n)</td>
<td>Sets the floating-point precision to n digits.</td>
</tr>
</tbody>
</table>

To use predefined manipulators, you must include the file iomanip.h in your program.

You can define your own manipulators. There are two basic types of manipulator:

- **Plain manipulator**—Takes an istream&, ostream&, or ios& argument, operates on the stream, and then returns its argument.
- **Parameterized manipulator**—Takes an istream&, ostream&, or ios& argument, one additional argument (the parameter), operates on the stream, and then returns its stream argument.
14.7.1 Using Plain Manipulators

A plain manipulator is a function that:

- Takes a reference to a stream
- Operates on it in some way
- Returns its argument

The shift operators taking (a pointer to) such a function are predefined for `iostreams`, so the function can be put in a sequence of input or output operators. The shift operator calls the function rather than trying to read or write a value. An example of a `tab` manipulator that inserts a `tab` in an `ostream` is:

```cpp
ostream& tab(ostream& os) {
    return os << '\t';
}
```

...  
`cout << x << tab << y;`

This is an elaborate way to achieve the following:

```cpp
const char tab = '\t';
...
cout << x << tab << y;
```

The following code is another example, which cannot be accomplished with a simple constant. Suppose you want to turn whitespace skipping on and off for an input stream. You can use separate calls to `ios::setf` and `ios::unsetf` to turn the `skipws` flag on and off, or you could define two manipulators.

```cpp
#include <iostream.h>
#include <iomanip.h>

istream& skipon(istream &is) {
    is.setf(ios::skipws, ios::skipws);
    return is;
}

istream& skipoff(istream& is) {
    is.unsetf(ios::skipws);
    return is;
}
...
int main ()
{
    int x,y;
    cin >> skipon >> x >> skipoff >> y;
    return 1;
}
```
### 14.7.2 Parameterized Manipulators

One of the parameterized manipulators that is included in `iomanip.h` is `setfill`. `setfill` sets the character that is used to fill out field widths. It is implemented as shown in the following example:

```cpp
//file setfill.cc
#include<iostream.h>
#include<iomanip.h>

//the private manipulator
static ios& sfill(ios & i, int f) {
    i.fill(f);
    return i;
}

//the public applicator
smanip_int setfill(int f) {
    return smanip_int(sfill, f);
}
```

A parameterized manipulator is implemented in two parts:

- **The manipulator.** It takes an extra parameter. In the previous code example, it takes an extra `int` parameter. You cannot place this manipulator function in a sequence of input or output operations, since there is no shift operator defined for it. Instead, you must use an auxiliary function, the applicator.

- **The applicator.** It calls the manipulator. The applicator is a global function, and you make a prototype for it available in a header file. Usually the manipulator is a static function in the file containing the source code for the applicator. The manipulator is called only by the applicator, and if you make it static, you keep its name out of the global address space.

Several classes are defined in the header file `iomanip.h`. Each class holds the address of a manipulator function and the value of one parameter. The `iomanip` classes are described in the man page `manip(3CC4)`. The previous example uses the `smanip_int` class, which works with an `ios`. Because it works with an `ios`, it also works with an `istream` and an `ostream`. The previous example also uses a second parameter of type `int`.

The applicator creates and returns a class object. In the previous code example the class object is an `smanip_int`, and it contains the manipulator and the `int` argument to the applicator. The `iomanip.h` header file defines the shift operators for this class. When the applicator function `setfill` appears in a sequence of input or output operations, the applicator function is called, and it returns a class. The shift operator acts on the class to call the manipulator function with its parameter value, which is stored in the class.

In the following example, the manipulator `print_hex`:

- Puts the output stream into the hex mode.
14.8 Strstreams: iostreams for Arrays

See the strstream(3CC4) man page.

14.9 Stdiobufs: iostreams for stdio Files

See the stdiobuf(3CC4) man page.

14.10 Streambufs

iostreams are the formatting part of a two-part (input or output) system. The other part of the system is made up of streambufs, which deal in input or output of unformatted streams of characters.

You usually use streambufs through iostreams, so you don’t have to worry about the details of streambufs. You can use streambufs directly if you choose to, for example, if you need to improve efficiency or to get around the error handling or formatting built into iostreams.

14.10.1 Working With Streambufs

A streambuf consists of a stream or sequence of characters and one or two pointers into that sequence. Each pointer points between two characters. (Pointers cannot actually point between characters, but it is helpful to think of them that way.) There are two kinds of streambuf pointers:
14.10.1 Streambufs

- A put pointer, which points just before the position where the next character will be stored
- A get pointer, which points just before the next character to be fetched

A streambuf can have one or both of these pointers.

14.10.1.1 Position of Pointers

The positions of the pointers and the contents of the sequences can be manipulated in various ways. Whether or not both pointers move when manipulated depends on the kind of streambuf used. Generally, with queue-like streambufs, the get and put pointers move independently; with file-like streambufs the get and put pointers always move together. A stream is an example of a queue-like stream; an fstream is an example of a file-like stream.

14.10.2 Using Streambufs

You never create an actual streambuf object, but only objects of classes derived from class streambuf. Examples are filebuf and strstreambuf, which are described in man pages filebuf(3CC4) and sdbuf(3), respectively. Advanced users may want to derive their own classes from streambuf to provide an interface to a special device or to provide other than basic buffering. Man pages sbufpub(3CC4) and sbufprot(3CC4) discuss how to do this.

Apart from creating your own special kind of streambuf, you may want to access the streambuf associated with an iostream to access the public member functions, as described in the man pages referenced above. In addition, each iostream has a defined inserter and extractor which takes a streambuf pointer. When a streambuf is inserted or extracted, the entire stream is copied.

Here is another way to do the file copy discussed earlier, with the error checking omitted for clarity:

```cpp
#include <fstream>

int main() {
  std::ifstream fromFile("thisFile");
  std::ofstream toFile("thatFile");
  toFile << fromFile.rdbuf();
}
```

We open the input and output files as before. Every iostream class has a member function rdbuf that returns a pointer to the streambuf object associated with it. In the case of an fstream, the streambuf object is type filebuf. The entire file associated with fromFile is copied (inserted into) the file associated with toFile. The last line could also be written like this:

```cpp
fromFile >> toFile.rdbuf();
```

The source file is then extracted into the destination. The two methods are entirely equivalent.
14.11 `iostream` Man Pages

A number of C++ man pages give details of the `iostream` library. The following table gives an overview of what is in each man page.

To access a classic `iostream` library man page, type:

```
example% man -s 3CC4 name
```

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>filebuf</td>
<td>Details the public interface for the class <code>filebuf</code>, which is derived from <code>streambuf</code> and is specialized for use with files. See the <code>sbufpub(3CC4)</code> and <code>sbufprot(3CC4)</code> man pages for details of features inherited from class <code>streambuf</code>. Use the <code>filebuf</code> class through class <code>fstream</code>.</td>
</tr>
<tr>
<td>fstream</td>
<td>Details specialized member functions of classes <code>ifstream</code>, <code>ofstream</code>, and <code>fstream</code>, which are specialized versions of <code>istream</code>, <code>ostream</code>, and <code>iostream</code> for use with files.</td>
</tr>
<tr>
<td>ios</td>
<td>Details parts of class <code>ios</code>, which functions as a base class for <code>iostreams</code>. It contains state data common to all streams.</td>
</tr>
<tr>
<td>ios.intro</td>
<td>Gives an introduction to and overview of <code>iostreams</code>.</td>
</tr>
</tbody>
</table>
| istream  | Details the following:  
  - Member functions for class `istream`, which supports interpretation of characters fetched from a `streambuf`  
  - Input formatting  
  - Positioning functions described as part of class `ostream`.  
  - Some related functions  
  - Related manipulators |
| manip    | Describes the input and output manipulators defined in the `iostream` library. |
| ostream  | Details the following:  
  - Member functions for class `ostream`, which supports interpretation of characters written to a `streambuf`  
  - Output formatting  
  - Positioning functions described as part of class `ostream`  
  - Some related functions  
  - Related manipulators |
14.12 **iostream Terminology**

The *iostream* library descriptions often use terms similar to terms from general programming, but with specialized meanings. The following table defines these terms as they are used in discussing the *iostream* library.

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbufprot</td>
<td>Describes the interface needed by programmers who are coding a class derived from class <em>streambuf</em>. Also refer to the sbufpub(3CC4) man page because some public functions are not discussed in the sbufprot(3CC4) man page.</td>
</tr>
<tr>
<td>sbufpub</td>
<td>Details the public interface of class <em>streambuf</em>, in particular, the public member functions of <em>streambuf</em>. This man page contains the information needed to manipulate a <em>streambuf</em>-type object directly, or to find out about functions that classes derived from <em>streambuf</em> inherit from it. If you want to derive a class from <em>streambuf</em>, also see the sbufprot(3CC4) man page.</td>
</tr>
<tr>
<td>ssbuf</td>
<td>Details the specialized public interface of class <em>strstreambuf</em>, which is derived from <em>streambuf</em> and specialized for dealing with arrays of characters. See the sbufpub(3CC4) man page for details of features inherited from class <em>streambuf</em>.</td>
</tr>
<tr>
<td>stdiobuf</td>
<td>Contains a minimal description of class <em>stdiobuf</em>, which is derived from <em>streambuf</em> and specialized for dealing with <em>stdio</em> FILES. See the sbufpub(3CC4) man page for details of features inherited from class <em>streambuf</em>.</td>
</tr>
<tr>
<td>strstream</td>
<td>Details the specialized member functions of <em>strstreams</em>, which are implemented by a set of classes derived from the <em>iostream</em> classes and specialized for dealing with arrays of characters.</td>
</tr>
</tbody>
</table>
### Table 14-4: iostream Terminology

<table>
<thead>
<tr>
<th>iostream Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>A word with two meanings, one specific to the iostream package and one more generally applied to input and output. When referring specifically to the iostream library, a buffer is an object of the type defined by the class streambuf. A buffer, generally, is a block of memory used to make efficient transfer of characters for input of output. With buffered I/O, the actual transfer of characters is delayed until the buffer is full or forcibly flushed. An unbuffered buffer refers to a streambuf where there is no buffer in the general sense defined above. This chapter avoids use of the term buffer to refer to streambufs. However, the man pages and other C++ documentation do use the term buffer to mean streambufs.</td>
</tr>
<tr>
<td>Extraction</td>
<td>The process of taking input from an iostream.</td>
</tr>
<tr>
<td>Fstream</td>
<td>An input or output stream specialized for use with files. Refers specifically to a class derived from class iostream when printed in courier font.</td>
</tr>
<tr>
<td>Insertion</td>
<td>The process of sending output into an iostream.</td>
</tr>
<tr>
<td>iostream</td>
<td>Generally, an input or output stream.</td>
</tr>
<tr>
<td>iostream library</td>
<td>The library implemented by the include files iostream.h, fstream.h, strstream.h, iomanip.h, and stdiostream.h. Because iostream is an object-oriented library, you should extend it. So, some of what you can do with the iostream library is not implemented.</td>
</tr>
<tr>
<td>Stream</td>
<td>An iostream, fstream, strstream, or user-defined stream in general.</td>
</tr>
<tr>
<td>Streambuf</td>
<td>A buffer that contains a sequence of characters with a put or get pointer, or both. When printed in courier font, it means the particular class. Otherwise, it refers generally to any object of class streambuf or a class derived from streambuf. Any stream object contains an object, or a pointer to an object, of a type derived from streambuf.</td>
</tr>
<tr>
<td>Strstream</td>
<td>An iostream specialized for use with character arrays. It refers to the specific class when printed in courier font.</td>
</tr>
</tbody>
</table>
Complex numbers are numbers made up of a real part and an imaginary part. For example:

\[ 3.2 + 4i \]
\[ 1 + 3i \]
\[ 1 + 2.3i \]

In the degenerate case, \( 0 + 3i \) is an entirely imaginary number generally written as \( 3i \), and \( 5 + 0i \) is an entirely real number generally written as \( 5 \). You can represent complex numbers using the \texttt{complex} data type.

\textbf{Note} – The complex arithmetic library (\texttt{libcomplex}) is available only for compatibility mode (\texttt{-compat =4}). In standard mode (the default mode), complex number classes with similar functionality are included with the C++ Standard Library \texttt{libCstd}.

\section*{15.1 The Complex Library}

The complex arithmetic library implements a complex number data type as a new data type and provides:

- Operators
- Mathematical functions (defined for the built-in numerical types)
- Extensions (for iostreams that allow input and output of complex numbers)
- Error handling mechanisms

Complex numbers can also be represented as an absolute value (or magnitude) and an argument (or angle). The library provides functions to convert between the real and imaginary (Cartesian) representation and the magnitude and angle (polar) representation.

The complex conjugate of a number has the opposite sign in its imaginary part.
15.1.1 Using the Complex Library

To use the complex library, include the header file complex.h in your program, and compile and link with the -library=complex option.

15.2 Type complex

The complex arithmetic library defines one class: class complex. An object of class complex can hold a single complex number. The complex number is constructed of two parts:

- The real part
- The imaginary part

```cpp
class complex {
    double re, im;
};
```

The value of an object of class complex is a pair of double values. The first value represents the real part; the second value represents the imaginary part.

15.2.1 Constructors of Class complex

There are two constructors for complex. Their definitions are:

```cpp
complex::complex() {re=0.0; im=0.0;}
complex::complex(double r, double i = 0.0) {re=r; im=i;}
```

If you declare a complex variable without specifying parameters, the first constructor is used and the variable is initialized, so that both parts are 0. The following example creates a complex variable whose real and imaginary parts are both 0:

```cpp
complex aComp;
```

You can give either one or two parameters. In either case, the second constructor is used. When you give only one parameter, that parameter is taken as the value for the real part and the imaginary part is set to 0. For example:

```cpp
complex aComp(4.533);
```

creates a complex variable with the following value:

`4.533 + 0i`

If you give two values, the first value is taken as the value of the real part and the second as the value of the imaginary part. For example:
complex aComp(8.999, 2.333);
creates a complex variable with the following value:

8.999 + 2.333i

You can also create a complex number using the `polar` function, which is provided in the complex arithmetic library (see “15.3 Mathematical Functions” on page 196). The `polar` function creates a complex value given the polar coordinates magnitude and angle.

There is no destructor for type `complex`.

### 15.2.2 Arithmetic Operators

The complex arithmetic library defines all the basic arithmetic operators. Specifically, the following operators work in the usual way and with the usual precedence:

+ - / * =

The subtraction operator (-) has its usual binary and unary meanings.

In addition, you can use the following operators in the usual way:

- Addition assign operator (+=)
- Subtraction assign operator (-=)
- Multiplication assign operator (*=)
- Division assign operator (/=)

However, the preceding four operators do not produce values that you can use in expressions. For example, the following expressions do not work:

```cpp
complex a, b;
...
if ((a+=2)==0) {...}; // illegal
b = a *= b; // illegal
```

You can also use the equality operator (==) and the inequality operator (!=) in their regular meaning.

When you mix real and complex numbers in an arithmetic expression, C++ uses the complex operator function and converts the real values to complex values.
15.3 Mathematical Functions

The complex arithmetic library provides a number of mathematical functions. Some are peculiar to complex numbers; the rest are complex-number versions of functions in the standard C mathematical library.

All of these functions produce a result for every possible argument. If a function cannot produce a mathematically acceptable result, it calls `complex_error` and returns some suitable value. In particular, the functions try to avoid actual overflow and call `complex_error` with a message instead. The following tables describe the remainder of the complex arithmetic library functions.

**Note** - The implementation of the `sqrt` and `atan2` functions is aligned with the C99 `csqrt` Annex G specification.

### TABLE 15–1 Complex Arithmetic Library Functions

<table>
<thead>
<tr>
<th>Complex Arithmetic Library Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>double abs(const complex)</code></td>
<td>Returns the magnitude of a complex number.</td>
</tr>
<tr>
<td><code>double arg(const complex)</code></td>
<td>Returns the angle of a complex number.</td>
</tr>
<tr>
<td><code>complex conj(const complex)</code></td>
<td>Returns the complex conjugate of its argument.</td>
</tr>
<tr>
<td><code>double imag(const complex&amp;)</code></td>
<td>Returns the imaginary part of a complex number.</td>
</tr>
<tr>
<td><code>double norm(const complex)</code></td>
<td>Returns the square of the magnitude of its argument. Faster than abs, but more likely to cause an overflow. For comparing magnitudes.</td>
</tr>
<tr>
<td><code>complex polar(double mag, double ang=0.0)</code></td>
<td>Takes a pair of polar coordinates that represent the magnitude and angle of a complex number and returns the corresponding complex number.</td>
</tr>
<tr>
<td><code>double real(const complex&amp;)</code></td>
<td>Returns the real part of a complex number.</td>
</tr>
</tbody>
</table>

### TABLE 15–2 Complex Mathematical and Trigonometric Functions

<table>
<thead>
<tr>
<th>Complex Arithmetic Library Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>complex acos(const complex)</code></td>
<td>Returns the angle whose cosine is its argument.</td>
</tr>
</tbody>
</table>
### 15.4 Error Handling

The complex library has these definitions for error handling:

```c
extern int errno;
class c_exception {...};
int complex_error(c_exception&);
```
The external variable errno is the global error state from the C library. errno can take on the values listed in the standard header <errno.h> (see the man page perror(3)). No function sets errno to zero, but many functions set it to other values.

To determine whether a particular operation fails:
1. Set errno to zero before the operation.
2. Test the operation.

The function complex_error takes a reference to type c_exception and is called by the following complex arithmetic library functions:

- exp
- log
- log10
- sinh
- cosh

The default version of complex_error returns zero. This return of zero means that the default error handling takes place. You can provide your own replacement function complex_error that performs other error handling. Error handling is described in the man page cplxerr(3CC4).

Default error handling is described in the man pages cplxtrig(3CC4) and cplxexp(3CC4). It is also summarized in the following table.

<table>
<thead>
<tr>
<th>Complex Arithmetic Library Function</th>
<th>Default Error Handling Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp</td>
<td>If overflow occurs, sets errno to ERANGE and returns a huge complex number.</td>
</tr>
<tr>
<td>log, log10</td>
<td>If the argument is zero, sets errno to EDOM and returns a huge complex number.</td>
</tr>
<tr>
<td>sinh, cosh</td>
<td>If the imaginary part of the argument causes overflow, returns a complex zero. If the real part causes overflow, returns a huge complex number. In either case, sets errno to ERANGE.</td>
</tr>
</tbody>
</table>

### 15.5 Input and Output

The complex arithmetic library provides default extractors and inserters for complex numbers, as shown in the following example:

```cpp
ostream& operator<<(ostream&, const complex&); // inserter
istream& operator>>(istream&, complex&);   // extractor
```
For basic information on extractors and inserters, see “14.2 Basic Structure of iostream Interaction” on page 172 and “14.3.1 Output Using iostream” on page 173.

For input, the complex extractor >> extracts a pair of numbers (surrounded by parentheses and separated by a comma) from the input stream and reads them into a complex object. The first number is taken as the value of the real part; the second as the value of the imaginary part. For example, given the declaration and input statement:

```cpp
complex x;
cin >> x;
```

and the input (3.45, 5), the value of x is equivalent to 3.45 + 5.0i. The reverse is true for inserters. Given `complex x(3.45, 5)`, `cout<<x` prints (3.45, 5).

The input usually consists of a pair of numbers in parentheses separated by a comma; white space is optional. If you provide a single number, with or without parentheses and white space, the extractor sets the imaginary part of the number to zero. Do not include the symbol `i` in the input text.

The inserter inserts the values of the real and imaginary parts enclosed in parentheses and separated by a comma. It does not include the symbol `i`. The two values are treated as doubles.

### 15.6 Mixed-Mode Arithmetic

Type `complex` is designed to fit in with the built-in arithmetic types in mixed-mode expressions. Arithmetic types are silently converted to type `complex`, and there are `complex` versions of the arithmetic operators and most mathematical functions. For example:

```cpp
int i, j;
double x, y;
complex a, b;
a = sin((b+i)/y) + x/j;
```

The expression `b+i` is mixed-mode. Integer `i` is converted to type `complex` via the constructor `complex::complex(double, double=0)`, the integer first being converted to type `double`. The result is to be divided by `y`, a `double`, so `y` is also converted to `complex` and the complex divide operation is used. The quotient is thus type `complex`, so the complex sine routine is called, yielding another `complex` result, and so on.

Not all arithmetic operations and conversions are implicit, or even defined, however. For example, complex numbers are not well-ordered, mathematically speaking, and complex numbers can be compared for equality only.

```cpp
complex a, b;
a == b; // OK
a != b; // OK
```
a < b; // error: operator < cannot be applied to type complex
a >= b; // error: operator >= cannot be applied to type complex

Similarly, there is no automatic conversion from type complex to any other type, because the concept is not well-defined. You can specify whether you want the real part, imaginary part, or magnitude, for example.

complex a;
double f(double);
f(abs(a)); // OK
f(a); // error: no match for f(complex)

15.7 Efficiency

The design of the complex class addresses efficiency concerns.

The simplest functions are declared inline to eliminate function call overhead.

Several overloaded versions of functions are provided when that makes a difference. For example, the pow function has versions that take exponents of type double and int as well as complex, since the computations for the former are much simpler.

The standard C math library header math.h is included automatically when you include complex.h. The C++ overloading rules then result in efficient evaluation of expressions like this:

double x;
complex x = sqrt(x);

In this example, the standard math function sqrt(double) is called, and the result is converted to type complex, rather than converting to type complex first and then calling sqrt(complex). This result falls right out of the overload resolution rules, and is precisely the result you want.

15.8 Complex Man Pages

The remaining documentation of the complex arithmetic library consists of the man pages listed in the following table.

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>cplx.intro(3CC4)</td>
<td>General introduction to the complex arithmetic library</td>
</tr>
</tbody>
</table>
### TABLE 15-3 Man Pages for Type complex (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartpol(3CC4)</td>
<td>Cartesian and polar functions</td>
</tr>
<tr>
<td>cxnorm(3CC4)</td>
<td>Error-handling functions</td>
</tr>
<tr>
<td>cplxexp(3CC4)</td>
<td>Exponential, log, and square root functions</td>
</tr>
<tr>
<td>cplxops(3CC4)</td>
<td>Arithmetic operator functions</td>
</tr>
<tr>
<td>cplxtrig(3CC4)</td>
<td>Trigonometric functions</td>
</tr>
</tbody>
</table>
16.1 Understanding Libraries

Libraries provide two benefits. First, they provide a way to share code among several applications. If you have such code, you can create a library with it and link the library with any application that needs it. Second, libraries provide a way to reduce the complexity of very large applications. Such applications can build and maintain relatively independent portions as libraries and so reduce the burden on programmers working on other portions.

Building a library simply means creating .o files (by compiling your code with the -c option) and combining the .o files into a library using the CC command. You can build two kinds of libraries, static (archive) libraries and dynamic (shared) libraries.

With static (archive) libraries, objects within the library are linked into the program’s executable file at link time. Only those .o files from the library that are needed by the application are linked into the executable. The name of a static (archive) library generally ends with a .a suffix.

With dynamic (shared) libraries, objects within the library are not linked into the program’s executable file, but rather the linker notes in the executable that the program depends on the library. When the program is executed, the system loads the dynamic libraries that the program requires. If two programs that use the same dynamic library execute at the same time, the operating system shares the library among the programs. The name of a dynamic (shared) library ends with a .so suffix.

Linking dynamically with shared libraries has several advantages over linking statically with archive libraries:

- The size of the executable is smaller.
Significant portions of code can be shared among programs at runtime, reducing the amount of memory use.

The library can be replaced at runtime without relinking with the application. (This is the primary mechanism that enables programs to take advantage of many improvements in the Solaris operating system without requiring relinking and redistribution of programs.)

The shared library can be loaded at runtime, using the `dlopen()` function call.

However, dynamic libraries have some disadvantages:

- Runtime linking has an execution-time cost.
- Distributing a program that uses dynamic libraries might require simultaneous distribution of the libraries it uses.
- Moving a shared library to a different location can prevent the system from finding the library and executing the program. (The environment variable `LD_LIBRARY_PATH` helps overcome this problem.)

### 16.2 Building Static (Archive) Libraries

The mechanism for building static (archive) libraries is similar to that of building an executable. A collection of object (.o) files can be combined into a single library using the `-xar` option of `CC`.

You should build static (archive) libraries using `CC -xar` instead of using the `ar` command directly. The C++ language generally requires that the compiler maintain more information than can be accommodated with traditional .o files, particularly template instances. The `-xar` option ensures that all necessary information, including template instances, is included in the library. You might not be able to accomplish this in a normal programming environment since `make` might not know which template files are actually created and referenced. Without `CC -xar`, referenced template instances might not be included in the library, as required. For example:

```bash
% CC -c foo.cc # Compile main file, templates objects are created.
% CC -xar -o foo.a foo.o # Gather all objects into a library.
```

The `-xar` flag causes `CC` to create a static (archive) library. The `-o` directive is required to name the newly created library. The compiler examines the object files on the command line, cross-references the object files with those known to the template repository, and adds those templates required by the user’s object files (along with the main object files themselves) to the archive.

Note – Use the `-xar` flag for creating or updating an existing archive only. Do not use it to maintain an archive. The `-xar` option is equivalent to `ar -cr`. 

---

Sun Studio 12 Update 1: C++ User’s Guide •
It is a good idea to have only one function in each .o file. If you are linking with an archive, an entire .o file from the archive is linked into your application when a symbol is needed from that particular .o file. Having one function in each .o file ensures that only those symbols needed by the application will be linked from the archive.

16.3 Building Dynamic (Shared) Libraries

Dynamic (shared) libraries are built the same way as static (archive) libraries, except that you use -G instead of -xar on the command line.

You should not use ld directly. As with static libraries, the CC command ensures that all the necessary template instances from the template repository are included in the library if you are using templates. All static constructors in a dynamic library that is linked to an application are called before main() is executed and all static destructors are called after main() exits. If a shared library is opened using dlopen(), all static constructors are executed at dlopen() and all static destructors are executed at dlclose().

You should use CC -G to build a dynamic library. When you use ld (the link-editor) or cc (the C compiler) to build a dynamic library, exceptions might not work and the global variables that are defined in the library are not initialized.

To build a dynamic (shared) library, you must create relocatable object files by compiling each object with the -Kpic or -KPIC option of CC. You can then build a dynamic library with these relocatable object files. If you get any bizarre link failures, you might have forgotten to compile some objects with -Kpic or -KPIC.

To build a C++ dynamic library named libfoo.so that contains objects from source files lsx1.cc and lsx2.cc, type:

```bash
% CC -G -o libfoo.so -h libfoo.so -Kpic lsx1.cc lsx2.cc
```

The -G option specifies the construction of a dynamic library. The -o option specifies the file name for the library. The -h option specifies an internal name for the shared library. The -Kpic option specifies that the object files are to be position-independent.

The CC -G command does not pass any -l options to the linker, ld. To ensure proper initialization order, a shared library must have an explicit dependency on each other shared library it needs. To create the dependencies, use a -l option for each such library. Typical C++ shared libraries will use one of the following sets of options:

```bash
-Lstd -lCrun -lc
-Llibrary=stlport4 -lCrun -lc
```

To be sure you have listed all needed dependencies, build the library with the -zdefs option. The linker will issue an error message for each missing symbol definition. To provide the missing definitions, add a -l option for those libraries.
To find out if you have included unneeded dependencies, use the commands

```
ldd -u -r mylib.so
ldd -U -r mylib.so
```

You can then rebuild `mylib.so` without the unneeded dependencies.

### 16.4 Building Shared Libraries That Contain Exceptions

Never use `-Bsymbolic` with programs containing C++ code, use linker map files instead. With `-Bsymbolic`, references in different modules can bind to different copies of what is supposed to be one global object.

The exception mechanism relies on comparing addresses. If you have two copies of something, their addresses won’t compare equal, and the exception mechanism can fail because the exception mechanism relies on comparing what are supposed to be unique addresses.

### 16.5 Building Libraries for Private Use

When an organization builds a library for internal use only, the library can be built with options that are not advised for more general use. In particular, the library need not comply with the system’s application binary interface (ABI). For example, the library can be compiled with the `-fast` option to improve its performance on a known architecture. Likewise, it can be compiled with the `-xregs=float` option to improve performance.

### 16.6 Building Libraries for Public Use

When an organization builds a library for use by other organizations, the management of the libraries, platform generality, and other issues become significant. A simple test for whether or not a library is public is to ask if the application programmer can recompile the library easily. Public libraries should be built in conformance with the system’s application binary interface (ABI). In general, this means that any processor-specific options should be avoided. (For example, do not use `- fast` or `-xtarget`.)

The SPARC ABI reserves some registers exclusively for applications. For V7 and V8, these registers are %g2, %g3, and %g4. For V9, these registers are %g2 and %g3. Since most compilations are for applications, the C++ compiler, by default, uses these registers for scratch registers, improving program performance. However, use of these registers in a public library is generally not compliant with the SPARC ABI. When building a library for public use, compile all objects with the `-xregs=no%appl` option to ensure that the application registers are not used.
16.7  Building a Library That Has a C API

If you want to build a library that is written in C++ but that can be used with a C program, you must create a C API (application programming interface). To do this, make all the exported functions `extern "C"`. Note that this can be done only for global functions and not for member functions.

If a C-interface library needs C++ run-time support and you are linking with `cc`, then you must also link your application with either `libC` (compatibility mode) or `libCrun` (standard mode) when you use the C-interface library. (If the C-interface library does not need C++ run-time support, then you do not have to link with `libC` or `libCrun`.) The steps for linking differ for archived and shared libraries.

When providing an archived C-interface library, you must provide instructions on how to use the library:

- If the C-interface library was built with `CC` in *standard mode* (the default), add `-lCrun` to the `cc` command line when using the C-interface library.
- If the C-interface library was built with `CC` in *compatibility mode* (`-compat`), add `-lC` to the `cc` command line when using the C-interface library.

When providing a shared C-interface library you must create a dependency on `libC` or `libCrun` at the time that you build the library. When the shared library has the correct dependency, you do not need to add `-lC` or `-lCrun` to the command line when you use the library.

- If you are building the C-interface library in *compatibility mode* (`-compat`), add `-lC` to the `CC` command line when you build the library.
- If you are building the C-interface library in *standard mode* (the default), add `-lCrun` to the `CC` command line when you build the library.

If you want to remove any dependency on the C++ runtime libraries, you should enforce the following coding rules in your library sources:

- Do not use any form of `new` or `delete` unless you provide your own corresponding versions.
- Do not use exceptions.
- Do not use runtime type information (RTTI).

16.8  Using `dlopen` to Access a C++ Library From a C Program

If you want to use `dlopen()` to open a C++ shared library from a C program, make sure that the shared library has a dependency on the appropriate C++ runtime (`libC.so.5` for `-compat=4`, or `libCrun.so.1` for `-compat=5`).

To do this, add `-lC` for `-compat=4` or add `-lCrun` for `-compat=5` to the command line when building the shared library. For example:
If the shared library uses exceptions and does not have a dependency on the C++ runtime library, your C program might behave erratically.
PART IV

Appendixes
This appendix details the command-line options for the C++ compiler. The features described apply to all platforms except as noted; features that are unique to the Solaris OS on SPARC-based systems are identified as SPARC, and the features that are unique to the Solaris and Linux OS on x86-based systems are identified as x86. Features limited to the Solaris OS only are marked Solaris; features limited only to Linux OS are marked Linux. Note that references to the Solaris OS imply the OpenSolaris OS as well.

The typographical conventions that are listed in the Preface are used in this section of the manual to describe individual options.

Parentheses, braces, brackets, pipe characters, and ellipses are metacharacters used in the descriptions of the options and are not part of the options themselves.

### A.1 How Option Information Is Organized

To help you find information, compiler option descriptions are separated into the following subsections. If the option is one that is replaced by or identical to some other option, see the description of the other option for full details.

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option Definition</td>
<td>A short definition immediately follows each option. (There is no heading for this category.)</td>
</tr>
<tr>
<td>Values</td>
<td>If the option has one or more values, this section defines each value.</td>
</tr>
</tbody>
</table>
TABLE A–1  Option Subsections  (Continued)

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defaults</td>
<td>If the option has a primary or secondary default value, it is stated here. The primary default is the option value in effect if the option is not specified. For example, if (-\text{compat}) is not specified, the default is (-\text{compat}=5). The secondary default is the option in effect if the option is specified, but no value is given. For example, if (-\text{compat}) is specified without a value, the default is (-\text{compat}=4).</td>
</tr>
<tr>
<td>Expansions</td>
<td>If the option has a macro expansion, it is shown in this section.</td>
</tr>
<tr>
<td>Examples</td>
<td>If an example is needed to illustrate the option, it is given here.</td>
</tr>
<tr>
<td>Interactions</td>
<td>If the option interacts with other options, the relationship is discussed here.</td>
</tr>
<tr>
<td>Warnings</td>
<td>If there are cautions regarding use of the option, they are noted here, as are actions that might cause unexpected behavior.</td>
</tr>
<tr>
<td>See also</td>
<td>This section contains references to further information in other options or documents.</td>
</tr>
<tr>
<td>&quot;Replace with&quot; &quot;Same as&quot;</td>
<td>If an option has become obsolete and has been replaced by another option, the replacement option is noted here. Options described this way may not be supported in future releases. If there are two options with the same general meaning and purpose, the preferred option is referenced here. For example, &quot;Same as (-\text{x0})&quot; indicates that (-\text{x0}) is the preferred option.</td>
</tr>
</tbody>
</table>

A.2 Option Reference

The following section alphabetically lists all the C++ compiler options and indicates any platform restrictions.

A.2.1  –386

x86: Same as \(-\text{target}=386\). This option is provided for backward compatibility only.

A.2.2  –486

x86: Same as \(-\text{target}=486\). This option is provided for backward compatibility only.
### A.2.3 

**–a**

Same as –xa.

### A.2.4

**–B binding**

Specifies whether a library binding for linking is **symbolic**, **dynamic** (shared), or **static** (nonshared).

You can use the –B option several times on a command line. This option is passed to the linker, `ld`.

**Note** - Many system libraries are only available as dynamic libraries in the Solaris 64-bit compilation environment. Therefore, do not use `-B static` as the last toggle on the command line.

#### A.2.4.1 Values

`binding` must be one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic</td>
<td>Directs the link editor to look for <code>liblib.so</code> (shared) files, and if they are not found, to look for <code>liblib.a</code> (static, nonshared) files. Use this option if you want shared library bindings for linking.</td>
</tr>
<tr>
<td>static</td>
<td>Directs the link editor to look only for <code>liblib.a</code> (static, nonshared) files. Use this option if you want nonshared library bindings for linking.</td>
</tr>
<tr>
<td>symbolic</td>
<td>Forces symbols to be resolved within a shared library if possible, even when a symbol is already defined elsewhere. See the <code>ld(1)</code> man page.</td>
</tr>
</tbody>
</table>

(No space is allowed between –B and the `binding` value.)

**Defaults**

If `-B` is not specified, –B`dynamic` is assumed.

**Interactions**

To link the C++ default libraries statically, use the –staticlib option.
The `-Bstatic` and `-Bdynamic` options affect the linking of the libraries that are provided by default. To ensure that the default libraries are linked dynamically, the last use of `-B` should be `-Bdynamic`.

In a 64-bit environment, many system libraries are available only as shared dynamic libraries. These include `libm.so` and `libc.so` (`libm.a` and `libc.a` are not provided). As a result, `-Bstatic` and `-dn` may cause linking errors in 64-bit Solaris operating systems. Applications must link with the dynamic libraries in these cases.

**Examples**
The following compiler command links `libfoo.a` even if `libfoo.so` exists; all other libraries are linked dynamically:

```
example% CC a.o -Bstatic -lfoo -Bdynamic
```

**Warnings**
Never use `-Bsymbolic` with programs containing C++ code, use linker map files instead.

With `-Bsymbolic`, references in different modules can bind to different copies of what is supposed to be one global object.

The exception mechanism relies on comparing addresses. If you have two copies of something, their addresses won’t compare equal, and the exception mechanism can fail because the exception mechanism relies on comparing what are supposed to be unique addresses.

If you compile and link in separate steps and are using the `-Bbinding` option, you must include the option in the link step.

**See also**

### A.2.5

#### C

Compile only; produce object `.o` files, but suppress linking.

This option directs the `CC` driver to suppress linking with `ld` and produce a `.o` file for each source file. If you specify only one source file on the command line, then you can explicitly name the object file with the `-o` option.

#### A.2.5.1

**Examples**

If you enter `CC -c x.cc`, the `x.o` object file is generated.

If you enter `CC -c x.cc -o y.o`, the `y.o` object file is generated.
Warnings

When the compiler produces object code for an input file (.c, .i), the compiler always produces a .o file in the working directory. If you suppress the linking step, the .o files are not removed.

See also

–o filename, –xe

A.2.6

–cg{89 | 92}

Same as –xcg[89|92].

A.2.7

–compat[={4|5}]

Sets the major release compatibility mode of the compiler. This option controls the __SUNPRO_CC_COMPAT and __cplusplus macros.

The C++ compiler has two principal modes. The compatibility mode accepts ARM semantics and language defined by the 4.2 compiler. The standard mode accepts constructs according to the ANSI/ISO standard. These two modes are incompatible with each other because the ANSI/ISO standard forces significant, incompatible changes in name mangling, vtable layout, and other ABI details. These two modes are differentiated by the –compat option as shown in the following values.

A.2.7.1 Values

The –compat option can have the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>–compat=4</td>
<td>(Compatibility mode) Set language and binary compatibility to that of the 4.0.1, 4.1, and 4.2 compilers. Set the __cplusplus preprocessor macro to 1 and the __SUNPRO_CC_COMPAT preprocessor macro to 4.</td>
</tr>
<tr>
<td>–compat=5</td>
<td>(Standard mode) Set language and binary compatibility to ANSI/ISO standard mode. Set the __cplusplus preprocessor macro to 199711L and the __SUNPRO_CC_COMPAT preprocessor macro to 5.</td>
</tr>
</tbody>
</table>

Defaults

If the –compat option is not specified, –compat=5 is assumed.

If only –compat is specified, –compat=4 is assumed.
Interactions

You cannot use the standard libraries in compatibility mode (-compat[=4]).

Use of -compat[=4] with any of the following options is not supported.

- -Bsymbolic
- -features=[no%]strictdestrorder
- -features=[no%]tmplife
- -library=[no%]iostream
- -library=[no%]Cstd
- -library=[no%]Crun
- -library=[no%]rwtools7_std
- -xarch=native64, -xarch=generic64, -xarch=v9, -xarch=v9a, or -xarch=v9b

Use of -compat=5 with any of the following options is not supported.

- -Bsymbolic
- +e
- features=[no%]arraynew
- features=[no%]explicit
- features=[no%]namespace
- features=[no%]rtti
- library=[no%]complex
- library=[no%]libC
- -vdelx

Warnings

When building a shared library do not use -Bsymbolic.

See also

C++ Migration Guide

A.2.8 +d

Does not expand C++ inline functions.

Under the C++ language rules, a C++ inline function is a function for which one of the following statements is true.

- The function is defined using the inline keyword,
- The function is defined (not just declared) inside a class definition
- The function is a compiler-generated class member function

Under the C++ language rules, the compiler can choose whether actually to inline a call to an inline function. The C++ compiler inlines calls to an inline function unless:
- The function is too complex,
- The +d option is selected, or
- The -g option is selected

A.2.8.1 Examples

By default, the compiler may inline the functions f() and memf2() in the following code example. In addition, the class has a default compiler-generated constructor and destructor that the compiler may inline. When you use +d, the compiler will not inline f() and C::mf2(), the constructor, and the destructor.

```cpp
inline int f() {return 0;} // may be inlined
class C {
  int mf1(); // not inlined unless inline definition comes later
  int mf2() {return 0;} // may be inlined
};
```

Interactions

This option is automatically turned on when you specify –g, the debugging option.

The –g0 debugging option does not turn on +d.

The +d option has no effect on the automatic inlining that is performed when you use -xO4 or -xO5.

See also

- g0, -g

A.2.9 -Dname[=def]

Defines the macro symbol name to the preprocessor.

Using this option is equivalent to including a #define directive at the beginning of the source. You can use multiple -D options.

A.2.9.1 Values

The following table shows the predefined macros. You can use these values in such preprocessor conditionals as #ifdef.
### Table A–2  Predefined Macros

<table>
<thead>
<tr>
<th>Platform</th>
<th>Macro Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARC and x86</td>
<td>_ARRAYNEW</td>
<td>__ARRAYNEW is defined if the “array” forms of operators new and delete are enabled. See -features=[no%]arraynew for more information.</td>
</tr>
<tr>
<td></td>
<td>BUILTIN_VA_ARG_INCR</td>
<td>For the _builtin_alloca, _builtin_va_list, and _ builtin_va_arg_incr keywords in varargs.h, stdarg.h, and sys/varargs.h.</td>
</tr>
<tr>
<td></td>
<td><strong>DATE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>FILE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>LINE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>STDC</strong></td>
<td>Set to 0 (zero)</td>
</tr>
<tr>
<td></td>
<td>__SUNPRO_CC=0x510</td>
<td>The value of _SUNPRO_CC indicates the release number of the compiler</td>
</tr>
<tr>
<td></td>
<td>__SUNPRO_CC_COMPAT=4 or _SUNPRO_CC_COMPAT=5</td>
<td>See &quot;A.2.7 –compat[=4][5]&quot; on page 215</td>
</tr>
<tr>
<td></td>
<td><strong>TIME</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>__cplusplus</td>
<td></td>
</tr>
<tr>
<td></td>
<td>‘uname -s’‘uname -r</td>
<td>tr . ’</td>
</tr>
<tr>
<td></td>
<td>__unix</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_BOOL</td>
<td>_BOOL is defined if type bool is enabled. See -features=[no%]bool for more information.</td>
</tr>
<tr>
<td></td>
<td>__WCHAR_T</td>
<td></td>
</tr>
<tr>
<td>unix</td>
<td>Unix</td>
<td></td>
</tr>
<tr>
<td>SPARC</td>
<td>__SUN_PREFETCH=1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>__SunOS_OSversion_OSversion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>__SVR4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>__sparc</td>
<td></td>
</tr>
</tbody>
</table>
### TABLE A–2  Predefined Macros  (Continued)

<table>
<thead>
<tr>
<th>Platform</th>
<th>Macro Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>__sun</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sparc</td>
<td></td>
<td>See Interactions.</td>
</tr>
<tr>
<td>sun</td>
<td></td>
<td>See Interactions.</td>
</tr>
<tr>
<td>SPARC v9</td>
<td>__sparcv9</td>
<td>64-bit compilation modes only</td>
</tr>
<tr>
<td>x86</td>
<td>i386</td>
<td></td>
</tr>
<tr>
<td></td>
<td>linux</td>
<td></td>
</tr>
<tr>
<td></td>
<td>__i386</td>
<td></td>
</tr>
<tr>
<td></td>
<td>__and64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>gnu__linux</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>__i386</td>
<td>See Interactions.</td>
</tr>
<tr>
<td></td>
<td>__linux</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>linux</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>__x86_64</td>
<td></td>
</tr>
</tbody>
</table>

If you do not use `=def, name` is defined as 1.

**Interactions**

If `+p` is used, `sun`, `unix`, `sparc`, and `i386` are not defined.

**See also**

`–U`

### A.2.10  `–d{y|n}`

Allows or disallows dynamic libraries for the entire executable.

This option is passed to `ld`.

This option can appear only once on the command line.
A.2.10.1 Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-dy</td>
<td>Specifies dynamic linking in the link editor.</td>
</tr>
<tr>
<td>-dn</td>
<td>Specifies static linking in the link editor.</td>
</tr>
</tbody>
</table>

Defaults

If no -d option is specified, -dy is assumed.

Interactions

In a 64-bit environment, many system libraries are available only as shared dynamic libraries. These include \texttt{libm.so} and \texttt{libc.so} (\texttt{libm.a} and \texttt{libc.a} are not provided). As a result, -Bstatic and -dn may cause linking errors in 64-bit Solaris operating systems. Applications must link with the dynamic libraries in these cases.

Warnings

This option causes fatal errors if you use it in combination with dynamic libraries. Most system libraries are only available as dynamic libraries.

See also

\texttt{ld(1)}, \textit{Linker and Libraries Guide}

A.2.11 -dalign

(SPARC) -dalign is equivalent to -xmemalign=8s. See "A.2.151 -xmemalign=ab" on page 316 for more information.

This option is silently ignored on x86 platforms.

A.2.11.1 Warnings

If you compile one program unit with -dalign, compile all units of a program with -dalign, or you might get unexpected results.

A.2.12 -dryrun

Shows the subcommands built by driver, but does not compile.
This option directs the CC driver to show, but not execute, the subcommands constructed by the compilation driver.

**A.2.13**

**–E**

Runs the preprocessor on source files; does not compile.

Directs the CC driver to run only the preprocessor on C++ source files, and to send the result to stdout (standard output). No compilation is done; no .o files are generated.

This option causes preprocessor-type line number information to be included in the output.

**A.2.13.1 Examples**

This option is useful for determining the changes made by the preprocessor. For example, the following program, foo.cc, generates the output shown in “A.2.13.1 Examples” on page 221

**EXAMPLE A–1**  Preprocessor Example Program foo.cc

```c
#if __cplusplus < 199711L
int power(int, int);
#else
template <> int power(int, int);
#endif

int main () {
    int x;
    x = power(2, 10);
}
```

**EXAMPLE A–2**  Preprocessor Output of foo.cc Using -E Option

```c
example% CC -E foo.cc
#4 "foo.cc"
template <> int power (int, int);

int main () {
    int x;
    x = power (2, 10);
}
```

**Warnings**

Output of this option might not be usable as input to a C++ compilation if the code contains templates under the "definitions-separate" model.
A.2 Option Reference

See also

-P

A.2.14  
+e{0|1}

Controls virtual table generation in compatibility mode (- compat[4]). Invalid and ignored when in standard mode (the default mode).

A.2.14.1 Values

The +e option can have the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Suppresses the generation of virtual tables and creates external references to those that are needed.</td>
</tr>
<tr>
<td>1</td>
<td>Creates virtual tables for all defined classes with virtual functions.</td>
</tr>
</tbody>
</table>

Interactions

When you compile with this option, also use the -features=no%except option. Otherwise, the compiler generates virtual tables for internal types used in exception handling.

If template classes have virtual functions, ensuring that the compiler generates all needed virtual tables, but does not duplicate these tables, might not be possible.

See also

C++ Migration Guide

A.2.15  
-erroff[=t]

This command suppresses C++ compiler warning messages and has no effect on error messages. This option applies to all warning messages whether or not they have been designated by -errwarn to cause a non-zero exit status.

A.2.15.1 Values

-t is a comma-separated list that consists of one or more of the following: tag, no%tag, %all, %none. Order is important; for example, %all, no%tag suppresses all warning messages except tag. The following table lists the -erroff values:
The `-erroff` Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tag</code></td>
<td>Suppresses the warning message specified by this <code>tag</code>. You can display the <code>tag</code> for a message by using the <code>-errtags=yes</code> option.</td>
</tr>
<tr>
<td><code>no%tag</code></td>
<td>Enables the warning message specified by this <code>tag</code>.</td>
</tr>
<tr>
<td><code>%all</code></td>
<td>Suppresses all warning messages.</td>
</tr>
<tr>
<td><code>%none</code></td>
<td>Enables all warning messages (default).</td>
</tr>
</tbody>
</table>

Defaults

The default is `-erroff=%none`. Specifying `-erroff` is equivalent to specifying `-erroff=%all`.

Examples

For example, `-erroff=tag` suppresses the warning message specified by this `tag`. On the other hand, `-erroff=%all,no%tag` suppresses all warning messages except the messages identified by `tag`.

You can display the `tag` for a warning message by using the `-errtags=yes` option.

Warnings

Only warning messages from the C++ compiler front-end that display a `tag` when the `-errtags` option is used can be suppressed with the `-erroff` option.

See Also

- `-errtags`, `-errwarn`

A.2.16  [-errtags[=a]]

Displays the message `tag` for each warning message of the C++ compiler front-end that can be suppressed with the `-erroff` option or made a fatal warning with the `-errwarn` option.

A.2.16.1 Values and Defaults

`a` can be either `yes` or `no`. The default is `-errtags=no`. Specifying `-errtags` is equivalent to specifying `-errtags=yes`.

Warnings

Messages from the C++ compiler driver and other components of the compilation system do not have error tags, and cannot be suppressed with `-erroff` or made fatal with `-errwarn`. 
A.2 Option Reference

See Also
-erroff, -errwarn

A.2.17 -errwarn[=t]

Use -errwarn to cause the C++ compiler to exit with a failure status for the given warning messages.

A.2.17.1 Values

\( t \) is a comma-separated list that consists of one or more of the following: \textit{tag}, \textit{no\%tag}, \textit{\%all}, \textit{\%none}. Order is important; for example \textit{\%all, no\%tag} causes cc to exit with a fatal status if any warning except \textit{tag} is issued.

The following table details the -errwarn values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{tag}</td>
<td>Cause \textit{CC} to exit with a fatal status if the message specified by this \textit{tag} is issued as a warning message. Has no effect if \textit{tag} is not issued.</td>
</tr>
<tr>
<td>\textit{no%tag}</td>
<td>Prevent \textit{CC} from exiting with a fatal status if the message specified by \textit{tag} is issued only as a warning message. Has no effect if the message specified by \textit{tag} is not issued. Use this option to revert a warning message that was previously specified by this option with \textit{tag} or \textit{%all} from causing \textit{cc} to exit with a fatal status when issued as a warning message.</td>
</tr>
<tr>
<td>\textit{%all}</td>
<td>Cause \textit{CC} to exit with a fatal status if any warning messages are issued. \textit{%all} can be followed by \textit{no%tag} to exempt specific warning messages from this behavior.</td>
</tr>
<tr>
<td>\textit{%none}</td>
<td>Prevents any warning message from causing \textit{CC} to exit with a fatal status should any warning message be issued.</td>
</tr>
</tbody>
</table>

Defaults

The default is -errwarn=\%none. If you specify -errwarn alone, it is equivalent to -errwarn=\%all.

Warnings

Only warning messages from the C++ compiler front-end that display a tag when the -errtags option is used can be specified with the -errwarn option to cause the compiler to exit with a failure status.
The warning messages generated by the C++ compiler change from release to release as the compiler error checking improves and features are added. Code that compiles using `-errwarn=all` without error may not compile without error in the next release of the compiler.

**See Also**

- `erroff`, `-errtags`, `-xwe`

### A.2.18  `-fast`

This option is a macro that can be effectively used as a starting point for tuning an executable for maximum runtime performance. `-fast` is a macro that can change from one release of the compiler to the next and expands to options that are target platform specific. Use the `-dryrun` or `-xdryrun` option to examine the expansion of `-fast`, and incorporate the appropriate options of `-fast` into the ongoing process of tuning the executable.

This option is a macro that selects a combination of compilation options for optimum execution speed on the machine upon which the code is compiled.

#### A.2.18.1  Expansions

This option provides near maximum performance for many applications by expanding to the following compilation options.

<table>
<thead>
<tr>
<th>Option</th>
<th>SPARC</th>
<th>x86</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-fns</code></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>-fsimple=2</code></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>-nofstore</code></td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td><code>-xarch</code></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>-xbuiltin=all</code></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>-xcache</code></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>-xchip</code></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>-xlibmil</code></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>-xlibmopt</code></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>-xmemalign</code></td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>
TABLE A–5 The -fast Expansion (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>SPARC</th>
<th>x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>–xO5</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>-xregs=frameptr</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>-xtarget=native</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Interactions

The -fast macro expands into compilation options that may affect other specified options. For example, in the following command, the expansion of the -fast macro includes -xtarget=native which reverts -xarch to one of the 32-bit architecture options.

Incorrect:

```bash
example% CC -xarch=v9 -fast test.cc
```

Correct:

```bash
example% CC -fast -xarch=v9 test.cc
```

See the description for each option to determine possible interactions.

The code generation option, the optimization level, the optimization of built-in functions, and the use of inline template files can be overridden by subsequent options (see examples). The optimization level that you specify overrides a previously set optimization level.

The -fast option includes -fns-ftrap=%none; that is, this option turns off all trapping.

Examples

The following compiler command results in an optimization level of –xO3.

```bash
example% CC –fast –xO3
```

The following compiler command results in an optimization level of –xO5.

```bash
example% CC -xO3 –fast
```

Warnings

If you compile and link in separate steps, the -fast option must appear in both the compile command and the link command.

Object binaries compiled with the -fast option are not portable. For example, using the following command on an UltraSPARC III system generates a binary that will not execute on an UltraSPARC II system.
example% CC -fast test.cc

Do not use this option for programs that depend on IEEE standard floating-point arithmetic; different numerical results, premature program termination, or unexpected SIGFPE signals can occur.

In previous SPARC releases, the -fast macro expanded to -fsimple=1. Now it expands to -fsimple=2.

The expansion of -fast includes -D_MATHERR_ERRNO_DONTCARE.

To display the expansion of -fast on any platform, run the command CC –dryrun –fast

>CC -dryrun -fast
### command line files and options (expanded):
### -dryrun -xO5 -xarch=sparcvis2 -xcache=64/32/4:1024/64/4 \
-xchip=ultra3i -xmemalign=8s -fsimple=2 -fns=yes -ftrap=%none \
-xlibmil -xlibmopt -xbuiltin=%all -D__MATHERR_ERRNO_DONTCARE

See also

-fns, -fsimple, -ftrap=%none, -xlibmil, -nofstore, -xO5, -xlibmopt, -xtarget=native

### A.2.19 –features=a[,a...]###

Enables/disables various C++ language features named in a comma-separated list.

#### A.2.19.1 Values

In both compatibility mode (-compat[=4]) and standard mode (the default mode), a can have the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%all</td>
<td>All the -features options that are valid for the specified mode (compatibility mode or standard mode).</td>
</tr>
<tr>
<td>[no%]altspell</td>
<td>[Do not] Recognize alternative token spellings (for example, “and” for “&amp;&amp;”). The default is no%altspell in compatibility mode and altspell in standard mode.</td>
</tr>
<tr>
<td>[no%]anachronisms</td>
<td>[Do not] Allow anachronistic constructs. When disabled (that is, -features=no%anachronisms), no anachronistic constructs are allowed. The default is anachronisms.</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[no%]bool</td>
<td>[Do not] Allow the bool type and literals. When enabled, the macro _BOOL==1. When not enabled, the macro is not defined. The default is no%bool in compatibility mode and bool in standard mode.</td>
</tr>
<tr>
<td>[no%]conststrings</td>
<td>[Do not] Put literal strings in read-only memory. The default is no%conststrings in compatibility mode and conststrings in standard mode.</td>
</tr>
<tr>
<td>[no%]except</td>
<td>[Do not] Allow C++ exceptions. When C++ exceptions are disabled (that is, -features=no%except), a throw-specification on a function is accepted but ignored; the compiler does not generate exception code. Note that the keywords try, throw, and catch are always reserved. See &quot;8.3 Disabling Exceptions&quot; on page 102. The default is except.</td>
</tr>
<tr>
<td>[no%]export</td>
<td>[Do not] Recognize the keyword export. The default is no%export in compatibility mode and export in standard mode. This feature is not yet implemented, but the export keyword is recognized.</td>
</tr>
<tr>
<td>[no%]extensions</td>
<td>[Do not] allow nonstandard code that is commonly accepted by other C++ compilers. See Table 3-17 for an explanation of the invalid code that is accepted by the compiler when you use the -features=extensions option. The default is no%extensions.</td>
</tr>
<tr>
<td>[no%]iddollar</td>
<td>[Do not] Allow a $ symbol as a noninitial identifier character. The default is no%iddollar.</td>
</tr>
<tr>
<td>[no%]localfor</td>
<td>[Do not] Use new local-scope rules for the for statement. The default is no%localfor in compatibility mode and localfor in standard mode.</td>
</tr>
<tr>
<td>[no%]mutable</td>
<td>[Do not] Recognize the keyword mutable. The default is no%mutable in compatibility mode and mutable in standard mode.</td>
</tr>
<tr>
<td>[no%]split_init</td>
<td>[Do not] Put initializers for nonlocal static objects into individual functions. When you use -features=no%split_init, the compiler puts all the initializers in one function. Using -features=no%split_init minimizes code size at the possible expense of compile time. The default is split_init.</td>
</tr>
</tbody>
</table>
### TABLE A–6  The -features Values for Compatibility Mode and Standard Mode  
(Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="no%transitions" alt="no%transitions" /></td>
<td>[Do not] allow ARM language constructs that are problematic in standard C++ and that may cause the program to behave differently than expected or that may be rejected by future compilers. When you use -features=no%transitions, the compiler treats these as errors. When you use -features=transitions in standard mode, the compiler issues warnings about these constructs instead of error messages. When you use -features=transitions in compatibility mode (-compat[=4]), the compiler displays the warnings about these constructs only if +w or +w2 is specified. The following constructs are considered to be transition errors: redefining a template after it was used, omitting the typename directive when it is needed in a template definition, and implicitly declaring type int. The set of transition errors may change in a future release. The default is transitions.</td>
</tr>
<tr>
<td><img src="%none" alt="%none" /></td>
<td>Turn off all the features that can be turned off for the specified mode.</td>
</tr>
</tbody>
</table>

In standard mode (the default mode), a can have the following additional values.

### TABLE A–7  The -features Values for Standard Mode Only

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="no%strictdestroorder" alt="no%strictdestroorder" /></td>
<td>[Do not] Follow the requirements specified by the C++ standard regarding the order of the destruction of objects with static storage duration. The default is strictdestroorder.</td>
</tr>
<tr>
<td><img src="no%tmplrefstatic" alt="no%tmplrefstatic" /></td>
<td>[Do not] allow function templates to refer to dependent static functions or static function templates. The default is the standard conformant no%tmplrefstatic.</td>
</tr>
<tr>
<td><img src="no%tmplife" alt="no%tmplife" /></td>
<td>[Do not] Clean up the temporary objects that are created by an expression at the end of the full expression, as defined in the ANSI/ISO C++ Standard. (When -features=no%tmplife is in effect, most temporary objects are cleaned up at the end of their block.) The default is no%tmplife in compat=4 mode and tmplife in standard mode.</td>
</tr>
</tbody>
</table>

In compatibility mode (-compat[=4]), a can have the following additional values.
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>![no%]arraynew</td>
<td>[Do not] Recognize array forms of operator new and operator delete (for example, operator new [ ] (void*)). When enabled, the macro __ARRAYNEW=1. When not enabled, the macro is not defined. The default is ![no%]arraynew.</td>
</tr>
<tr>
<td>![no%]explicit</td>
<td>[Do not] Recognize the keyword explicit. The default is ![no%]explicit.</td>
</tr>
<tr>
<td>![no%]namespace</td>
<td>[Do not] Recognize the keywords namespace and using. The default is ![no%]namespace.</td>
</tr>
<tr>
<td>![no%]rtti</td>
<td>[Do not] Allow runtime type information (RTTI). RTTI must be enabled to use the <code>dynamic_cast&lt;&gt;</code> and <code>typeid</code> operators. For <code>-compat=4</code> mode, the default is ![no%]rtti. Otherwise, the default is ![no%]rtti, and the option <code>-features=rtti</code> is not allowed.</td>
</tr>
</tbody>
</table>

**Note** – The ![no%]castop setting is allowed for compatibility with makefiles written for the C++ 4.2 compiler, but has no effect on later compiler versions. The new style casts (`const_cast`, `dynamic_cast`, `reinterpret_cast`, and `static_cast`) are always recognized and cannot be disabled.

**Defaults**

If `-features` is not specified, the default in compatibility mode (`-compat=[4]`) is:

```
-features=%none,anachronisms,except,split_init,transitions
```

In the default "standard mode",

```
-features=%all,no%extensions,no%iddollar,no%tmplrefstatic
```

is assumed.

**Interactions**

This option accumulates instead of overrides.

Use of the following in standard mode (the default) is not compatible with the standard libraries and headers:
In compatibility mode (\texttt{\textasciitilde compat\{=4\}}), the \texttt{-features=transitions} option has no effect unless you specify the \texttt{+w} option or the \texttt{+w2} option.

**Warnings**

Be careful when you specify \texttt{-features=%all} or \texttt{-features=%none}. The set of features can change with each compiler release and with each patch. Consequently, you can get unintended behavior.

The behavior of a program might change when you use the \texttt{-features=tmplife} option. Testing whether the program works both with and without the \texttt{-features=tmplife} option is one way to test the program's portability.

The compiler assumes \texttt{-features=split\_init} by default in compat mode (\texttt{-compat\{=4\}}). If you use the \texttt{-features=%none} option to turn off other features, you may find it desirable to turn the splitting of initializers into separate functions back on by using \texttt{-features=%none,split\_init} instead.

**See also**

Table 3–17 and the C++ Migration Guide

**A.2.20** \texttt{-filt\{=filter[,filter...]\}}

Controls the filtering that the compiler normally applies to linker and compiler error messages.

**A.2.20.1** **Values**

\textit{filter} must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>{no}errors</td>
<td>[Do not] Show the C++ explanations of the linker error messages. The suppression of the explanations is useful when the linker diagnostics are provided directly to another tool.</td>
</tr>
<tr>
<td>{no}names</td>
<td>[Do not] Demangle the C++ mangled linker names.</td>
</tr>
</tbody>
</table>
### TABLE A-9  
**The -filt Values (Continued)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]returns</td>
<td>[Do not] Demangle the return types of functions. Suppression of this type of demangling helps you to identify function names more quickly, but note that in the case of co-variant returns some functions differ only in the return type.</td>
</tr>
<tr>
<td>[no%]stdlib</td>
<td>[Do not] Simplify names from the standard library in both the linker and compiler error messages. This makes it easier for you to recognize the names of standard library template types.</td>
</tr>
<tr>
<td>%all</td>
<td>Equivalent to <code>-filt=errors,names,returns,stdlib</code>. This is the default behavior.</td>
</tr>
<tr>
<td>%none</td>
<td>Equivalent to <code>-filt=no%errors,no%names,no%returns,no%stdlib</code>.</td>
</tr>
</tbody>
</table>

### Defaults

If you do not specify the `-filt` option, or if you specify `-filt` without any values, then the compiler assumes `-filt=%all.`

### Examples

The following examples show the effects of compiling this code with the `-filt` option.

```c++
// filt_demo.cc
class type {
public:
  virtual ~type(); // no definition provided
};

int main()
{
  type t;
}
```

When you compile the code without the `-filt` option, the compiler assumes `-filt=errors,names,returns,stdlib` and displays the standard output.

```
example% CC filt_demo.cc
Undefined first referenced
  symbol in file
type::~type() filt_demo.o
type::__vtbl filt_demo.o
[Hint: try checking whether the first non-inlined, non-pure virtual function of class type is defined]
ld: fatal: Symbol referencing errors. No output written to a.out
```
The following command suppresses the demangling of the C++ mangled linker names and suppresses the C++ explanations of linker errors.

```
example% CC -filt=no_names,no_errors filt_demo.cc
Undefined symbol first referenced in file
__icEType2T6M_v filt_demo.o
__icETypeG_vtbl filt_demo.o
ld: fatal: Symbol referencing errors. No output written to a.out
```

Now consider this code:

```cpp
#include <string>
#include <list>
int main()
{
    std::list<int> l;
    std::string s(l); // error here
}
```

Here's the output when you specify `--filt=no_stdlib`:

```
Error: Cannot use std::list<int, std::allocator<int>> to initialize std::basic_string<char, std::char_traits<char>, std::allocator<char>>.
```

Here's the output when you specify `--filt=stdlib`:

```
Error: Cannot use std::list<int> to initialize std::string .
```

**Interactions**

When you specify `no_names`, neither `returns` nor `no_returns` has an effect. That is, the following options are equivalent:

- `-filt=no_names`
- `-filt=no_names,no_returns`
- `-filt=no_names,returns`

**A.2.21 --flags**

Same as `--xhelp=flags`. 
A.2.22  
\texttt{-fma=\{none | fused\}}

(SPARC) Enables automatic generation of floating-point, fused, multiply-add instructions.  
\texttt{-fma=none} disables generation of these instructions. \texttt{-fma=fused} allows the compiler to attempt to find opportunities to improve the performance of the code by using floating-point, fused, multiply-add instructions.  

The default is \texttt{-fma=none}.  

The minimum requirements are \texttt{-xarch=sparc|fma} and an optimization level of at least \texttt{-xO2} for the compiler to generate fused multiply-add instructions. The compiler marks the binary program if fused multiply-add instructions are generated in order to prevent the program from executing on platforms that do not support them.  

Fused multiply-adds eliminate the intermediate rounding step between the multiply and the add. Consequently, programs may produce different results when compiled with \texttt{-fma=fused}, although precision will tend to be increased rather than decreased.

A.2.23  
\texttt{-fnonstd}

Causes hardware traps to be enabled for floating-point overflow, division by zero, and invalid operations exceptions. These results are converted into SIGFPE signals; if the program has no SIGFPE handler, it terminates with a memory dump (unless you limit the core dump size to 0).  

SPARC: In addition, \texttt{-fnonstd} selects SPARC nonstandard floating point.

A.2.23.1  
\textbf{Defaults}  
If \texttt{-fnonstd} is not specified, IEEE 754 floating-point arithmetic exceptions do not abort the program, and underflows are gradual.

\textbf{Expansions}  
\begin{itemize}  
\item \texttt{x86: -fnonstd} expands to \texttt{-ftrap=common}.  
\item \texttt{SPARC: -fnonstd} expands to \texttt{-fns -ftrap=common}.  
\end{itemize}  

\textbf{See also}  
\texttt{-fns, -ftrap=common, Numerical Computation Guide}.

A.2.24  
\texttt{-fns=\{yes | no\}}

- SPARC: Enables/disables the SPARC nonstandard floating-point mode.
-fns=yes (or -fns) causes the nonstandard floating point mode to be enabled when a program begins execution.

This option provides a way of toggling the use of nonstandard or standard floating-point mode following some other macro option that includes –fns, such as –fast.

On some SPARC devices, the nonstandard floating-point mode disables “gradual underflow,” causing tiny results to be flushed to zero rather than to produce subnormal numbers. It also causes subnormal operands to be silently replaced by zero.

On those SPARC devices that do not support gradual underflow and subnormal numbers in hardware, -fns=yes (or -fns) can significantly improve the performance of some programs.

- (x86) Selects SSE flush-to-zero mode and, where available, denormals-are-zero mode.

This option causes subnormal results to be flushed to zero. Where available, this option also causes subnormal operands to be treated as zero.

This option has no effect on traditional x86 floating-point operations that do not utilize the SSE or SSE2 instruction set.

### A.2.24.1 Values

The -fns option can have the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Selects nonstandard floating-point mode</td>
</tr>
<tr>
<td>no</td>
<td>Selects standard floating-point mode</td>
</tr>
</tbody>
</table>

#### Defaults

If -fns is not specified, the nonstandard floating point mode is not enabled automatically. Standard IEEE 754 floating-point computation takes place, that is, underflows are gradual.

If only –fns is specified, –fns=yes is assumed.

#### Examples

In the following example, -fast expands to several options, one of which is -fns=yes which selects nonstandard floating-point mode. The subsequent -fns=no option overrides the initial setting and selects floating-point mode.

```
example% CC foo.cc -fast -fns=no
```
**Warnings**

When nonstandard mode is enabled, floating-point arithmetic can produce results that do not conform to the requirements of the IEEE 754 standard.

If you compile one routine with the -fns option, then compile all routines of the program with the -fns option; otherwise, you might get unexpected results.

This option is effective only on SPARC devices, and only if used when compiling the main program. On x86 devices, the option is ignored.

Use of the -fns=yes (or -fns) option might generate the following message if your program experiences a floating-point error normally managed by the IEEE floating-point trap handlers:

**See also**

*Numerical Computation Guide, ieee_sun(3M)*

### A.2.25 -fprecision=p

x86: Sets the non-default floating-point precision mode.

The -fprecision option sets the rounding precision mode bits in the Floating Point Control Word. These bits control the precision to which the results of basic arithmetic operations (add, subtract, multiply, divide, and square root) are rounded.

#### A.2.25.1 Values

`p` must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>single</td>
<td>Rounds to an IEEE single-precision value.</td>
</tr>
<tr>
<td>double</td>
<td>Rounds to an IEEE double-precision value.</td>
</tr>
<tr>
<td>extended</td>
<td>Rounds to the maximum precision available.</td>
</tr>
</tbody>
</table>

If `p` is `single` or `double`, this option causes the rounding precision mode to be set to `single` or `double` precision, respectively, when a program begins execution. If `p` is `extended` or the -fprecision option is not used, the rounding precision mode remains at the `extended` precision.

The `single` precision rounding mode causes results to be rounded to 24 significant bits, and `double` precision rounding mode causes results to be rounded to 53 significant bits. In the default `extended` precision mode, results are rounded to 64 significant bits. This mode controls
only the precision to which results in registers are rounded, and it does not affect the range. All results in register are rounded using the full range of the extended double format. Results that are stored in memory are rounded to both the range and precision of the destination format, however.

The nominal precision of the `float` type is `single`. The nominal precision of the `long double` type is `extended`.

**Defaults**

When the `–fprecision` option is not specified, the rounding precision mode defaults to `extended`.

**Warnings**

This option is effective only on x86 devices and only if used when compiling the main program. On SPARC devices, this option is ignored.

**A.2.26**

`–fround=r`

Sets the IEEE rounding mode in effect at startup.

This option sets the IEEE 754 rounding mode that:

- Can be used by the compiler in evaluating constant expressions
- Is established at runtime during the program initialization

The meanings are the same as those for the `ieee_flags` subroutine, which can be used to change the mode at runtime.

**A.2.26.1 Values**

`r` must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>nearest</td>
<td>Rounds towards the nearest number and breaks ties to even numbers.</td>
</tr>
<tr>
<td>tozero</td>
<td>Rounds to zero.</td>
</tr>
<tr>
<td>negative</td>
<td>Rounds to negative infinity.</td>
</tr>
<tr>
<td>positive</td>
<td>Rounds to positive infinity.</td>
</tr>
</tbody>
</table>

**Defaults**

When the `–fround` option is not specified, the rounding mode defaults to `–fround=nearest`. 
Warnings

If you compile one routine with `-fround=r`, compile all routines of the program with the same `-fround=r` option; otherwise, you might get unexpected results.

This option is effective only if used when compiling the main program.

**A.2.27** `-fsimple[=n]`

Selects floating-point optimization preferences.

This option allows the optimizer to make simplifying assumptions concerning floating-point arithmetic.

**A.2.27.1 Values**

If `n` is present, it must be 0, 1, or 2.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Permit no simplifying assumptions. Preserve strict IEEE 754 conformance.</td>
</tr>
</tbody>
</table>
| 1     | Allow conservative simplification. The resulting code does not strictly conform to IEEE 754, but numeric results of most programs are unchanged. With `-fsimple=1`, the optimizer can assume the following:  
  - IEEE754 default rounding/trapping modes do not change after process initialization.  
  - Computation producing no visible result other than potential floating-point exceptions can be deleted.  
  - Computation with infinities or NaNs as operands needs to propagate NaNs to their results; that is, `x*0` can be replaced by 0.  
  - Computations do not depend on sign of zero. With `-fsimple=1`, the optimizer is not allowed to optimize completely without regard to roundoff or exceptions. In particular, a floating-point computation cannot be replaced by one that produces different results when rounding modes are held constant at runtime. |

---

A.2 Option Reference
TABLE A-13  The -f simple Values  (Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Includes all the functionality of -f simple=1 and also permits aggressive floating-point optimization that can cause many programs to produce different numeric results due to changes in rounding. For example, permit the optimizer to replace all computations of ( x/y ) in a given loop with ( x*z ), where ( x/y ) is guaranteed to be evaluated at least once in the loop ( z=1/y ), and the values of ( y ) and ( z ) are known to have constant values during execution of the loop.</td>
</tr>
</tbody>
</table>

Defaults

If -f simple is not designated, the compiler uses -f simple=0.

If -f simple is designated but no value is given for \( n \), the compiler uses -f simple=1.

Interactions

-fast implies -f simple=2.

Warnings

This option can break IEEE 754 conformance.

See also

-fast

*Techniques for Optimizing Applications: High Performance Computing* written by Rajat Garg and Ilya Sharapov for a more detailed explanation of how optimization can impact precision.

A.2.28  -f store

x86:

Forces precision of floating-point expressions.

This option causes the compiler to convert the value of a floating-point expression or function to the type on the left side of an assignment rather than leave the value in a register when the following is true:

- The expression or function is assigned to a variable.
- The expression is cast to a shorter floating-point type.

To turn off this option, use the -nof store option.
A.2.28.1 Warnings

Due to roundoffs and truncation, the results can be different from those that are generated from the register values.

See also

–nofstored

A.2.29

–ftrap=\([t, t...]\)

Sets the IEEE trapping mode in effect at startup but does not install a SIGFPE handler. You can use ieee_handler(3M) or fex_set_handler(3M) to simultaneously enable traps and install a SIGFPE handler. If you specify more than one value, the list is processed sequentially from left to right.

A.2.29.1 Values

\(i\) can be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>([\text{no%}])division</td>
<td>[Do not] Trap on division by zero.</td>
</tr>
<tr>
<td>([\text{no%}])inexact</td>
<td>[Do not] Trap on inexact result.</td>
</tr>
<tr>
<td>([\text{no%}])invalid</td>
<td>[Do not] Trap on invalid operation.</td>
</tr>
<tr>
<td>([\text{no%}])overflow</td>
<td>[Do not] Trap on overflow.</td>
</tr>
<tr>
<td>([\text{no%}])underflow</td>
<td>[Do not] Trap on underflow.</td>
</tr>
<tr>
<td>%all</td>
<td>Trap on all of the above.</td>
</tr>
<tr>
<td>%none</td>
<td>Trap on none of the above.</td>
</tr>
<tr>
<td>common</td>
<td>Trap on invalid, division by zero, and overflow.</td>
</tr>
</tbody>
</table>

Note that the \([\text{no\%}]\) form of the option is used only to modify the meaning of the \%all and common values, and must be used with one of these values, as shown in the example. The \([\text{no\%}]\) form of the option by itself does not explicitly cause a particular trap to be disabled.

 Defaults

If you do not specify \(-f\)trap, the compiler assumes \(-f\)trap=%none.
Examples

\(-ftrap=\%all, no\%inexact\) means to set all traps except inexact.

Warnings

If you compile one routine with \(-ftrap=t\), compile all routines of the program with the same \(-ftrap=t\) option; otherwise, you might get unexpected results.

Use the \(-ftrap=inexact\) trap with caution. Use of \(-ftrap=inexact\) results in the trap being issued whenever a floating-point value cannot be represented exactly. For example, the following statement generates this condition:

\[
x = 1.0 / 3.0;
\]

This option is effective only if used when compiling the main program. Be cautious when using this option. If you wish to enable the IEEE traps, use \(-ftrap=\text{common}\).

See also

ieee_handler(3M), fex_set_handling(3M) man pages.

A.2.30

\(-G\)

Build a dynamic shared library instead of an executable file.

All source files specified in the command line are compiled with \(-xcode=\text{pic13}\) by default.

When building a shared library that uses templates, it is necessary in most cases to include in the shared library those template functions that are instantiated in the template data base. Using this option automatically adds those templates to the shared library as needed.

If you are creating a shared object by specifying \(-G\) along with other compiler options that must be specified at both compile time and link time, make sure that those same options are also specified at both compile time and link time when you link with the resulting shared object.

When you create a shared object, all the object files compiled with \(-xarch=v9\), must also be compiled with an explicit \(-xcode\) value as recommended in "A.2.119 \(-xcode=a\) on page 292.

A.2.30.1

Interactions

The following options are passed to the linker if \(-c\) (the compile-only option) is not specified:

\[-dy\]
\[-G\]
\[-R\]
**Warnings**

Do not use `ld -G` to build shared libraries; use `CC -G`. The `CC` driver automatically passes several options to `ld` that are needed for C++.

When you use the `-G` option, the compiler does not pass any default `-l` options to `ld`. If you want the shared library to have a dependency on another shared library, you must pass the necessary `-l` option on the command line. For example, if you want the shared library to be dependent upon `libCrun`, you must pass `-lCrun` on the command line.

**See also**

`-dy`, `-Kpic`, `-xcode=pic13`, `-ztext`, `ld(1)` man page, “16.3 Building Dynamic (Shared) Libraries” on page 205.

**A.2.31**

`-g`

Produces additional symbol table information for debugging with `dbx(1)` or the Debugger and for analysis with the Performance Analyzer `analyzer(1)`.

Instructs both the compiler and the linker to prepare the file or program for debugging and for performance analysis.

The tasks include:

- Producing detailed information, known as `stabs`, in the symbol table of the object files and the executable
- Producing some “helper functions,” which the debugger can call to implement some of its features
- Disabling the inline generation of functions
- Disabling certain levels of optimization

**A.2.31.1 Interactions**

If you use this option with `-xOlevel` (or its equivalent options, such as `-O`), you will get limited debugging information. For more information, see “A.2.157 -xOlevel” on page 321.

If you use this option and the optimization level is `-xO4` or higher, the compiler provides best-effort symbolic information with full optimization.

When you specify this option, the `+d` option is specified automatically unless you also specify `-O` or `-xO`. 
Note – In previous releases, this option forced the compiler to use the incremental linker (ild) by default instead of the linker (ld) for link-only invocations of the compiler. That is, with -g, the compiler’s default behavior was to automatically invoke ild in place of ld whenever you used the compiler to link object files, unless you specified -G or source files on the command line. This is no longer the case. The incremental linker is no longer available.

To use the full capabilities of the Performance Analyzer, compile with the -g option. While some performance analysis features do not require -g, you must compile with -g to view annotated source, some function level information, and compiler commentary messages. See the analyzer(1) man page and “Compiling Your Program for Data Collection and Analysis” in Program Performance Analysis Tools for more information.

The commentary messages that are generated with -g describe the optimizations and transformations that the compiler made while compiling your program. Use the er_src(1) command to display the messages, which are interleaved with the source code.

Warnings

If you compile and link your program in separate steps, then including the -g option in one step and excluding it from the other step will not affect the correctness of the program, but it will affect the ability to debug the program. Any module that is not compiled with -g (or -g0), but is linked with -g (or -g0) will not be prepared properly for debugging. Note that compiling the module that contains the function main with the -g option (or the -g0 option) is usually necessary for debugging.

See also

+d, -g0, -xs, analyzer(1) man page, er_src(1) man page, ld(1) man page, Debugging a Program With dbx (for details about stabs), Program Performance Analysis Tools.

A.2.32 -g0

Compiles and links for debugging, but does not disable inlining.

This option is the same as -g, except that +d is disabled and dbx cannot step into inlined functions.

If you specify -g0 and the optimization level is -xO3 or lower, the compiler provides best-effort symbolic information with almost full optimization. Tail-call optimization and back-end inlining are disabled.

A.2.32.1 See also

+d, -g, Debugging a Program With dbx
A.2.33  

-H  

Prints path names of included files.

On the standard error output (stderr), this option prints, one per line, the path name of each #include file contained in the current compilation.

A.2.34  

-h[ ]name  

Assigns the name name to the generated dynamic shared library.

This is a linker option, passed to ld. In general, the name after -h should be exactly the same as the one after -o. A space between the -h and name is optional.

The compile-time loader assigns the specified name to the shared dynamic library you are creating. It records the name in the library file as the intrinsic name of the library. If there is no -h name option, then no intrinsic name is recorded in the library file.

Every executable file has a list of shared library files that are needed. When the runtime linker links the library into an executable file, the linker copies the intrinsic name from the library into that list of needed shared library files. If there is no intrinsic name of a shared library, then the linker copies the path of the shared library file instead.

When a shared library is built without the -h option, the runtime loader looks only for the file name of the library. You can replace the library with a different library with the same file name. If the shared library has an intrinsic name, the loader checks the intrinsic name when loading the file. If the intrinsic name does not match, the loader will not use the replacement file.

A.2.34.1 Examples

example% CC -G -o libx.so.1 -h libx.so.1 a.o b.o c.o

A.2.35  

-help  

Same as -xhelp=flags.

A.2.36  

-Ipathname  

Add pathname to the #include file search path.

This option adds pathname to the list of directories that are searched for #include files with relative file names (those that do not begin with a slash).

The compiler searches for quote-included files (of the form #include "foo.h") in this order.
1. In the directory containing the source
2. In the directories named with -I options, if any
3. In the include directories for compiler-provided C++ header files, ANSI C header files, and special-purpose files
4. In the /usr/include directory

The compiler searches for bracket-included files (of the form `#include <foo.h>`) in this order.

1. In the directories named with -I options, if any
2. In the include directories for compiler-provided C++ header files, ANSI C header files, and special-purpose files
3. In the /usr/include directory

**Note** – If the spelling matches the name of a standard header file, also refer to “12.7.5 Standard Header Implementation” on page 150.

### A.2.36.1 Interactions

The -I option allows you to override the default search rules.

If you specify `-library=no%Cstd`, then the compiler does not include in its search path the compiler-provided header files that are associated with the C++ standard libraries. See “12.7 Replacing the C++ Standard Library” on page 148.

If `–ptipath` is not used, the compiler looks for template files in `–Ipathname`.

Use `–Ipathname` instead of `–ptipath`.

This option accumulates instead of overrides.

**Warnings**

Never specify the compiler installation area, `/usr/include`, `/lib`, or `/usr/lib`, as search directories.

**See also**

- `–I`

### A.2.37 - I -

Change the include-file search rules to the following:

For include files of the form `#include "foo.h"`, search the directories in the following order:

1. The directories named with -I options (both before and after -I-).
2. The directories for compiler-provided C++ header files, ANSI C header files, and special-purpose files.

3. The /usr/include directory.

For include files of the form `#include <foo.h>`, search the directories in the following order:

1. The directories named in the `-I` options that appear after `-I`.

2. The directories for compiler-provided C++ header files, ANSI C header files, and special-purpose files.

3. The /usr/include directory.

**Note** – If the name of the include file matches the name of a standard header, also refer to "12.7.5 Standard Header Implementation" on page 150.

### A.2.37.1 Examples

The following example shows the results of using `-I` when compiling `prog.cc`.

```c
prog.cc
#include "a.h"
#include <b.h>
#include "c.h"
#include "c.h"
#ifndef _C_H_1
#define _C_H_1
int c1;
#endif
inc/a.h
#ifndef _A_H
#define _A_H
#include "c.h"
int a;
#endif
inc/b.h
#ifndef _B_H
#define _B_H
#include <c.h>
int b;
#endif
inc/c.h
#ifndef _C_H_2
#define _C_H_2
int c2;
#endif
```
The following command shows the default behavior of searching the current directory (the directory of the including file) for include statements of the form `#include "foo.h"`. When processing the `#include "c.h"` statement in `inc/a.h`, the compiler includes the `c.h` header file from the `inc` subdirectory. When processing the `#include "c.h"` statement in `prog.cc`, the compiler includes the `c.h` file from the directory containing `prog.cc`. Note that the `-H` option instructs the compiler to print the paths of the included files.

```
example% CC -c -Iinc -H prog.cc
inc/a.h
    inc/c.h
inc/b.h
    inc/c.h
c.h
```

The next command shows the effect of the `-I-` option. The compiler does not look in the including directory first when it processes statements of the form `#include "foo.h"`. Instead, it searches the directories named by the `-I` options in the order that they appear in the command line. When processing the `#include "c.h"` statement in `inc/a.h`, the compiler includes the `.h` header file instead of the `inc/c.h` header file.

```
example% CC -c -I. -I- -Iinc -H prog.cc
inc/a.h
    ./c.h
inc/b.h
    ./c.h
    inc/c.h
    ./c.h
```

**Interactions**

When `-I-` appears in the command line, the compiler never searches the current directory, unless the directory is listed explicitly in a `-I` directive. This effect applies even for include statements of the form `#include "foo.h"`.

**Warnings**

Only the first `-I-` in a command line causes the described behavior.

Never specify the compiler installation area, `/usr/include`, `/lib`, or `/usr/lib`, as search directories.

**A.2.38**

```
-i
```

Tells the linker, `ld`, to ignore any `LD_LIBRARY_PATH` setting.
A.2.39 **-include filename**

This option causes the compiler to treat *filename* as if it appears in the first line of a primary source file as a `#include` preprocessor directive. Consider the source file `t.c`:

```c
main()
{
  ...
}
```

If you compile `t.c` with the command `cc -include t.h t.c`, the compilation proceeds as if the source file contains the following:

```c
#include "t.h"
main()
{
  ...
}
```

The first directory the compiler searches for *filename* is the current working directory and not the directory containing the main source file, as is the case when a file is explicitly included. For example, the following directory structure contains two header files with the same name, but at different locations:

```
foo/
  t.c
  t.h
bar/
  u.c
  t.h
```

If your working directory is `foo/bar` and you compile with the command `cc ../*.c -include t.h`, the compiler includes `t.h` from `foo/bar`, not `foo/` as would be the case with a `#include` directive from within the source file `t.c`.

If the compiler cannot find the file specified with `-include` in the current working directory, it searches the normal directory paths for the file. If you specify multiple `-include` options, the files are included in the order they appear on the command line.

A.2.40 **-inline**

Same as `-xinline`.

A.2.41 **-instances=a**

Controls the placement and linkage of template instances.
A.2.41.1 **Values**

A must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| `extern` | Places all needed instances into the template repository within comdat sections and gives them global linkage. (If an instance in the repository is out of date, it is reinstantiated.)

  **Note:** If you are compiling and linking in separate steps and you specify `-instances=extern` for the compilation step, you must also specify it for the link step. |
| `explicit` | Places explicitly instantiated instances into the current object file and gives them global linkage. Does not generate any other needed instances. |
| `global` | Places all needed instances into the current object file and gives them global linkage. |
| `semiexplicit` | Places explicitly instantiated instances into the current object file and gives them global linkage. Places all instances needed by the explicit instances into the current object file and gives them global linkage. Does not generate any other needed instances. |
| `static` | **Note:** `-instances=static` is deprecated. There is no longer any reason to use `-instances=static`, because `-instances=global` now gives you all the advantages of static without the disadvantages. This option was provided in earlier compilers to overcome problems that do not exist in this version of the compiler.

  Places all needed instances into the current object file and gives them static linkage. |

**Defaults**

If `-instances` is not specified, `-instances=global` is assumed.

**See also**

"7.2.4 Template Instance Placement and Linkage" on page 92.

A.2.42 **–instlib=filename**

Use this option to inhibit the generation of template instances that are duplicated in a library, either shared or static, and the current object. In general, if your program shares large numbers of instances with libraries, try `-instlib=filename` and see whether compilation time improves.
A.2.42.1 **Values:**

Use the *filename* argument to specify the library that you know contains the existing template instances. The filename argument must contain a forward slash '/' character. For paths relative to the current directory, use dot-slash './'.

**Defaults:**

The -instlib=filename option has no default and is only used if you specify it. This option can be specified multiple times and accumulates.

**Example:**

Assume that the libfoo.a and libbar.so libraries instantiate many template instances that are shared with your source file a.cc. Adding -instlib=filename and specifying the libraries helps reduce compile time by avoiding the redundancy.

```
example% CC -c -instlib=./libfoo.a -instlib=./libbar.so a.cc
```

**Interactions:**

When you compile with -g, if the library specified with -instlib=file is not compiled with -g, those template instances will not be debuggable. The workaround is to avoid -instlib=file when you use -g.

**Warning**

If you specify a library with -instlib, you must link with that library.

**See Also:**

- template, -instances, -pti

A.2.43 **–KPIC**

SPARC: Same as -xcode=pic32.

x86: Same as -Kpic.

Use this option to compile source files when building a shared library. Each reference to a global datum is generated as a dereference of a pointer in the global offset table. Each function call is generated in pc-relative addressing mode through a procedure linkage table.
A.2.44  
–Kpic

SPARC: Same as –xcode=pic13.

x86: Compiles with position-independent code.

Use this option to compile source files when building a shared library. Each reference to a global datum is generated as a dereference of a pointer in the global offset table. Each function call is generated in pc-relative addressing mode through a procedure linkage table.

A.2.45  
–keeptmp

Retains temporary files created during compilation.

Along with –verbose=diags, this option is useful for debugging.

A.2.45.1  See also

–v, –verbose

A.2.46  
–Lpath

Adds path to list of directories to search for libraries.

This option is passed to ld. The directory that is named by path is searched before compiler-provided directories.

A.2.46.1  Interactions

This option accumulates instead of overrides.

Warnings

Never specify the compiler installation area, /usr/include, /lib, or /usr/lib, as search directories.

A.2.47  
–lib

Adds library liblib.a or liblib.so to the linker’s list of search libraries.

This option is passed to ld. Normal libraries have names such as liblib.a or liblib.so, where the lib and .a or .so parts are required. You should specify the lib part with this option. Put as many libraries as you want on a single command line; they are searched in the order specified with –Ldir.
A.2.47.1 Interactions
This option accumulates instead of overrides.
It is always safer to put \texttt{-lx} after the list of sources and objects to insure that libraries are searched in the correct order.

Warnings
To ensure proper library linking order, you must use \texttt{-mt}, rather than \texttt{-\texttt{l}thread}, to link with \texttt{libthread}.

See also
\texttt{-Ldir}, \texttt{-mt}, “11.4.8 An Example Application” on page 133, and Tools.h++ Class Library Reference

A.2.48 \texttt{--libmieee}
Same as \texttt{-xlibmieee}.

A.2.49 \texttt{--libmil}
Same as \texttt{-xlibmil}.

A.2.50 \texttt{-library=[,\ldots]}
Incorporates specified CC-provided libraries into compilation and linking.

A.2.50.1 Values
For compatibility mode (\texttt{-compat [-4]}), \texttt{l} must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]f77</td>
<td>Deprecated. Use \texttt{-xlang=f77} instead.</td>
</tr>
<tr>
<td>[no%]f90</td>
<td>Deprecated. Use \texttt{-xlang=f90} instead.</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[no%]f95</td>
<td>Deprecated. Use -xlang=f95 instead.</td>
</tr>
<tr>
<td>[no%]rwtools7</td>
<td>[Do not] Use classic-iostreams Tools.h++ version 7.</td>
</tr>
<tr>
<td>[no%]rwtools7_dbg</td>
<td>[Do not] Use debug-enabled Tools.h++ version 7.</td>
</tr>
<tr>
<td>[no%]complex</td>
<td>[Do not] Use libcomplex for complex arithmetic.</td>
</tr>
<tr>
<td>[no%]interval</td>
<td>Deprecated. Do not use. Use -xia.</td>
</tr>
<tr>
<td>[no%]libc</td>
<td>[Do not] Use libc, the C++ support library.</td>
</tr>
<tr>
<td>[no%]gc</td>
<td>[Do not] Use libgc, garbage collection.</td>
</tr>
<tr>
<td>[no%]sunperf</td>
<td>[Do not] Use the Sun Performance Library™</td>
</tr>
<tr>
<td>%none</td>
<td>Use no C++ libraries except for libc.</td>
</tr>
</tbody>
</table>

For standard mode (the default mode), I must be one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]f77</td>
<td>Deprecated. Use -xlang=f77 instead.</td>
</tr>
<tr>
<td>[no%]f90</td>
<td>Deprecated. Use -xlang=f90 instead.</td>
</tr>
<tr>
<td>[no%]f95</td>
<td>Deprecated. Use -xlang=f95 instead.</td>
</tr>
<tr>
<td>[no%]rwtools7</td>
<td>[Do not] Use classic-iostreams Tools.h++ version 7.</td>
</tr>
<tr>
<td>[no%]rwtools7_dbg</td>
<td>[Do not] Use debug-enabled Tools.h++ version 7.</td>
</tr>
<tr>
<td>[no%]interval</td>
<td>Deprecated. Do not use. Use -xia.</td>
</tr>
<tr>
<td>[no%]iostream</td>
<td>[Do not] Use libiostream, the classic iostreams library.</td>
</tr>
<tr>
<td>[no%]Cstd</td>
<td>[Do not] Use libCstd, the C++ standard library. [Do not] Include the compiler-provided C++ standard library header files.</td>
</tr>
<tr>
<td>[no%]Crun</td>
<td>[Do not] Use libCrun, the C++ runtime library.</td>
</tr>
<tr>
<td>[no%]gc</td>
<td>[Do not] Use libgc, garbage collection.</td>
</tr>
</tbody>
</table>
The -library Values for Standard Mode (Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]stlport4</td>
<td>[Do not] Use STLport's Standard Library implementation version 4.5.3 instead of the default \libCstd. For more information about using STLport's implementation, see &quot;13.3 STLport&quot; on page 168.</td>
</tr>
<tr>
<td>[no%]stlport4_dbg</td>
<td>[Do not] Use STLport's debug-enabled library.</td>
</tr>
<tr>
<td>[no%]sunperf</td>
<td>[Do not] Use the Sun Performance Library.</td>
</tr>
<tr>
<td>%none</td>
<td>Use no C++ libraries, except for \libCrun.</td>
</tr>
</tbody>
</table>

Defaults

- **Compatibility mode** (--compat=[4])
  - If -library is not specified, -library=\libC is assumed.
  - The \libC library is always included unless it is specifically excluded using -library=no%\libC.

- **Standard mode (the default mode)**
  - The \libCstd library is always included unless it is specifically excluded using -library=%none or -library=no%\libCstd or -library=stlport4.
  - The \libCrun library always is included unless it is specifically excluded using -library=no%\libCrun.

Regardless of standard or compat mode, the \libm and \libc libraries are always included, even if you specify -library=%none.

Examples

To link in standard mode without any C++ libraries (except \libCrun), use:

```
example% CC -library=%none
```

To include the classic-iostreams Rogue Wave tools.h++ library in standard mode:

```
example% CC -library=rwtools7,iostream
```

To include the standard-iostreams Rogue Wave tools.h++ library in standard mode:

```
example% CC -library=rwtools7_std
```

To include the classic-iostreams Rogue Wave tools.h++ library in compatibility mode:

```
example% CC -compat -library=rwtools7
```
Interactions

If a library is specified with `-library`, the proper `–I` paths are set during compilation. The proper `–L`, `–Y P`, `–R` paths and `–l` options are set during linking.

This option accumulates instead of overrides.

When you use the interval arithmetic libraries, you must include one of the following libraries: `libC`, `libCstd`, or `libiostream`.

Use of the `-library` option ensures that the `-l` options for the specified libraries are emitted in the right order. For example, the `-l` options are passed to `ld` in the order `-lrwtool -liostream` for both `-library=rwtools7,iostream` and `-library=iostream,rwtools7`.

The specified libraries are linked before the system support libraries are linked.

You cannot use `-library=sunperf` and `-xlic_lib=sunperf` on the same command line.

You can use at most only one of `-library=stlport4`, or `-library=Cstd` options on any command line.

Only one Rogue Wave tools library can be used at a time and you cannot use any Rogue Wave tools library with `-library=stlport4`.

When you include the classic-iostreams Rogue Wave tools library in standard mode (the default mode), you must also include `libiostream` (see the C++ Migration Guide for additional information). You can use the standard-iostreams Rogue Wave tools library in standard mode only. The following command examples show both valid and invalid use of the Rogue Wave tools.h++ library options.

% CC -compat -library=rwtools7 foo.cc       <-- valid
% CC -compat -library=rwtools7_std foo.cc   <-- invalid
% CC -library=rwtools7,iostream foo.cc     <-- valid, classic iostreams
% CC -library=rwtools7 foo.cc              <-- invalid
% CC -library=rwtools7_std foo.cc          <-- valid, standard iostreams
% CC -library=rwtools7_std,iostream foo.cc <-- invalid

If you include both `libCstd` and `libiostream`, you must be careful to not use the old and new forms of iostreams (for example, `cout` and `std::cout`) within a program to access the same file. Mixing standard iostreams and classic iostreams in the same program is likely to cause problems if the same file is accessed from both classic and standard iostream code.

Compatibility-mode programs that do not link the `libC` library cannot use all features of the C++ language. Similarly, standard-mode programs that do not link `Crun` or any of the `Cstd`, or `stlport4` libraries, cannot use all features of the C++ language.

If `-xnolib` is specified, `-library` is ignored.
Warnings

If you compile and link in separate steps, the set of -library options that appear in the compile command must appear in the link command.

The stlport4, Cstd, and iostream libraries provide their own implementation of I/O streams. Specifying more than one of these with the -library option can result in undefined program behavior. For more information about using STLport’s implementation, see “13.3 STLport” on page 168.

The set of libraries is not stable and might change from release to release.

See also


For information on using the -library=no%cstd option to enable use of your own C++ standard library, see “12.7 Replacing the C++ Standard Library” on page 148.

A.2.51 -m32| -m64

Specifies the memory model for the compiled binary object.

Use -m32 to create 32-bit executables and shared libraries. Use -m64 to create 64-bit executables and shared libraries.

The ILP32 memory model (32-bit int, long, pointer data types) is the default on all Solaris platforms and on Linux platforms that are not 64-bit enabled. The LP64 memory model (64-bit long, pointer data types) is the default on Linux platforms that are 64-bit enabled. -m64 is permitted only on platforms that are enabled for the LP64 model.

Object files or libraries compiled with -m32 cannot be linked with object files or libraries compiled with -m64.

Modules that are compiled with -m32| -m64 must also be linked with -m32| -m64. For a complete list of compiler options that must be specified at both compile time and at link time, see “3.3.3 Compile-Time and Link-Time Options” on page 50.

When compiling applications with large amounts of static data on x64 platforms, using -m64, -xmodel=medium may also be required. Be aware that some Linux platforms do not support the medium model.
Note that in previous compiler releases, the memory model, ILP32 or LP64, was implied by the choice of the instruction set with -xarch. Starting with the Sun Studio 12 compilers, this is no longer the case. On most platforms, just adding -m64 to the command line is sufficient to create 64-bit objects.

On Solaris, -m32 is the default. On Linux systems supporting 64-bit programs, -m64 -xarch=sse2 is the default.

See also -xarch.

A.2.52 -mc

Removes duplicate strings from the .comment section of the object file. If the string contains blanks, the string must be enclosed in quotation marks. When you use the -mc option, the mcs -c command is invoked.

A.2.53 migration

Explains where to get information about migrating source code that was built for earlier versions of the compiler.

Note - This option might cease to exist in the next release.

A.2.54 -misalign

SPARC: Permits misaligned data, which would otherwise generate an error, in memory. This is shown in the following code:

```c
char b[100];
int f(int * ar) {
    return *(int *) (b +2) + *ar;
}
```

This option informs the compiler that some data in your program is not properly aligned. Thus, very conservative loads and stores must be used for any data that might be misaligned, that is, one byte at a time. Using this option may cause significant degradation in runtime performance. The amount of degradation is application dependent.

A.2.54.1 Interactions

When using #pragma pack on a SPARC platform to pack denser than the type’s default alignment, the -misalign option must be specified for both the compilation and the linking of the application.
Misaligned data is handled by a trap mechanism that is provided by `ld` at runtime. If an optimization flag (-x012345 or an equivalent flag) is used with the `-misalign` option, the additional instructions required for alignment of misaligned data are inserted into the resulting object file and will not generate runtime misalignment traps.

**Warnings**

If possible, do not link aligned and misaligned parts of the program.

If compilation and linking are performed in separate steps, the `-misalign` option must appear in both the compile and link commands.

**A.2.55**

-`mr[,,string]`

Removes all strings from the `.comment` section of the object file and, if `string` is supplied, places `string` in that section. If the string contains blanks, the string must be enclosed in quotation marks. When you use this option, the command `mcs -d [- a string]` is invoked.

**A.2.56**

-`mt`

Use this option to compile and link multithreaded code using the Solaris threads or POSIX threads API. The `-mt` option assures that libraries are linked in the appropriate order.

This option passes `-D_REENTRANT` to the preprocessor.

To use Solaris threads, include the `thread.h` header file and compile with the `-mt` option. To use POSIX threads on Solaris platforms, include the `pthread.h` header file and compile with the `-mt -lpthread` options.

On Linux platforms, only the POSIX threads API is available. (There is no `libthread` on Linux platforms.) Consequently, `-mt` on Linux platforms adds `-lpthread` instead of `-lthread`. To use POSIX threads on Linux platforms, compile with `-mt`.

Note that when compiling with `-G`, neither `-lthread` nor `-lpthread` are automatically included by `-mt`. You will need to explicitly list these libraries when building a shared library.

The `-xopenmp` option (for using the OpenMP shared-memory parallelization API) includes `-mt` automatically.

If you compile with `-mt` and link in a separate step, you must use the `-mt` option in the link step as well as the compile step. If you compile and link one translation unit with `-mt`, you must compile and link all units of the program with `-mt`.

**A.2.56.1**

**See Also**

A.2.57 –native
Same as –xtarget=native.

A.2.58 –noex
Same as –features=none_except.

A.2.59 –nofstore
x86: Disables forced precision of an expression.
This option does not force the value of a floating-point expression or function to the type on the
left side of an assignment, but leaves the value in a register when either of the following are true:
- The expression or function is assigned to a variable
  or
- The expression or function is cast to a shorter floating-point type

A.2.59.1 See also
–fstore

A.2.60 –nolib
Same as –xnolib.

A.2.61 –nolibmil
Same as –xnolibmil.

A.2.62 –noqueue
(Obsolete) Disables license queueing.
If no license is available, this option returns without queuing your request and without
compiling. A nonzero status is returned for testing makefiles. This option is obsolete and is
ignored.

A.2.63 –norunpath
Does not build a runtime search path for shared libraries into the executable.
If an executable file uses shared libraries, then the compiler normally builds in a path that points
the runtime linker to those shared libraries. To do so, the compiler passes the –R option to ld.
The path depends on the directory where you have installed the compiler.

This option is recommended for building executables that will be shipped to customers who
may have a different path for the shared libraries that are used by the program. Refer to “12.6
Using Shared Libraries” on page 148

A.2.63.1 Interactions

If you use any shared libraries under the compiler installed area and you also use –norunpath,
then you should either use the –R option at link time or set the environment variable
LD_LIBRARY_PATH at runtime to specify the location of the shared libraries. Doing so allows the
runtime linker to find the shared libraries.

A.2.64 –O

The -O macro now expands to -xO3 instead of -xO2.

The change in default yields higher run-time performance. However, -xO3 may be
inappropriate for programs that rely on all variables being automatically considered volatile.
Typical programs that might have this assumption are device drivers and older multi-threaded
applications that implement their own synchronization primitives. The work around is to
compile with -xO2 instead of -O.

A.2.65 –Olevel

Same as -xOlevel.

A.2.66 –o filename

Sets the name of the output file or the executable file to filename.

A.2.66.1 Interactions

When the compiler must store template instances, it stores them in the template repository in
the output file’s directory. For example, the following command writes the object file to
./sub/a.o and writes template instances into the repository contained within
./sub/SunWS_cache.

example% CC -instances=extern -o sub/a.o a.cc

The compiler reads from the template repositories corresponding to the object files that it reads.
For example, the following command reads from ./sub1/SunWS_Cache and
./sub2/SunWS_cache, and, if necessary, writes to ./SunWS_cache.
example% CC -instances=extern sub1/a.o sub2/b.o

For more information, see “7.4 The Template Repository” on page 96.

Warnings
The filename must have the appropriate suffix for the type of file to be produced by the compilation. It cannot be the same file as the source file, since the CC driver does not overwrite the source file.

A.2.67  +p
Ignore nonstandard preprocessor asserts.

A.2.67.1  Defaults
If +p is not present, the compiler recognizes nonstandard preprocessor asserts.

Interactions
If +p is used, the following macros are not defined:
■ sun
■ unix
■ sparc
■ i386

A.2.68  –P
Only preprocesses source; does not compile. (Outputs a file with a .i suffix.)
This option does not include preprocessor-type line number information in the output.

A.2.68.1  See also
–E

A.2.69  –p
Obsolete, see “A.2.165 –xpg” on page 331.

A.2.70  –pentium
x86: Replace with –xtarget=pentium.
A.2.71  
\texttt{-pg}  
Same as \texttt{-xpg}.

A.2.72  
\texttt{-PIC}  
SPARC: Same as \texttt{-xcode=pic32}.  
x86: Same as \texttt{-Kpic}.

A.2.73  
\texttt{-pic}  
SPARC: Same as \texttt{-xcode=pic13}.  
x86: Same as \texttt{-Kpic}.

A.2.74  
\texttt{-pta}  
Same as \texttt{-template=wholeclass}.

A.2.75  
\texttt{-ptipath}  
Specifies an additional search directory for template source.  
This option is an alternative to the normal search path set by \texttt{-Ipathname}. If the \texttt{-ptipath} option is used, the compiler looks for template definition files on this path and ignores the \texttt{-Ipathname} option.  
Using the \texttt{-Ipathname} option instead of \texttt{-ptipath} produces less confusion.

A.2.75.1  \textbf{Interactions}  
This option accumulates instead of overrides.  

\textbf{See also}  
\texttt{-Ipathname}, and "7.5.2 Definitions Search Path" on page 98

A.2.76  
\texttt{-pto}  
Same as \texttt{-instances=static}.  

**A.2.77** 

`-ptr`

This option is obsolete and is ignored by the compiler.

**A.2.77.1 Warnings**

Even though the `-ptr` option is ignored, you should remove `-ptr` from all compilation commands because, in a later release, it may be reused with a different behavior.

**See also**

For information about repository directories, see “7.4 The Template Repository” on page 96.

**A.2.78** 

`-ptv`

Same as `-verbose=template`.

**A.2.79** 

`-Qoption phase option[,option...]`

Passes `option` to the compilation `phase`.

To pass multiple options, specify them in order as a comma-separated list. Options that are passed to components with `-Q` might be reordered. Options that the driver recognizes are kept in the correct order. Do not use `-Q` for options that the driver already recognizes. For example, the C++ compiler recognizes the `-z` option for the linker (`ld`). If you issue a command like this

```
CC -G -zallextract mylib.a -zdefaultextract ... // correct
```

the `-z` options are passed in order to the linker. But if you specify the command like this

```
CC -G -Qoption ld -zallextract mylib.a -Qoption ld -zdefaultextract ... // error
```

the `-z` options can be reordered, giving incorrect results.

**A.2.79.1 Values**

`phase` must have one of the following values.

<table>
<thead>
<tr>
<th>TABLE A–18</th>
<th>The <code>-Qoption</code> Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARC</td>
<td>x86</td>
</tr>
<tr>
<td>ccfe</td>
<td>ccfe</td>
</tr>
</tbody>
</table>
TABLE A-18  The -Qoption Values  (Continued)

<table>
<thead>
<tr>
<th>SPARC</th>
<th>x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>iropt</td>
<td>iropt</td>
</tr>
<tr>
<td>cg</td>
<td>ube</td>
</tr>
<tr>
<td>CClk</td>
<td>CClk</td>
</tr>
<tr>
<td>ld</td>
<td>ld</td>
</tr>
<tr>
<td>—</td>
<td>ir2hf</td>
</tr>
<tr>
<td>fbe</td>
<td>fbe</td>
</tr>
</tbody>
</table>

**Examples**

In the following command line, when ld is invoked by the CC driver, -Qoption passes the -i and -m options to ld.

```
example% CC -Qoption ld -i,-m test.c
```

**Warnings**

Be careful to avoid unintended effects. For example,

```
-Qoption ccfe -features=bool,iddollar
```

is interpreted as

```
-Qoption ccfe -features=bool -Qoption ccfe iddollar
```

The correct usage is

```
-Qoption ccfe -features=bool,-features=iddollar
```

**A.2.80**  
-`qoption phase option`

Same as -Qoption.

**A.2.81**  
-`qp`

Same as -p.
A.2.82  

–Qproduce sourcetype

Causes the CC driver to produce output of the type sourcetype.

Sourcetype suffixes are defined below.

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.i</td>
<td>Preprocessed C++ source from ccfe</td>
</tr>
<tr>
<td>.o</td>
<td>Object file the code generator</td>
</tr>
<tr>
<td>.s</td>
<td>Assembler source from cg</td>
</tr>
</tbody>
</table>

A.2.83  

–qproduce sourcetype

Same as –Qproduce.

A.2.84  

–Rpathname[:pathname…]

Builds dynamic library search paths into the executable file.

This option is passed to \d.

A.2.84.1 Defaults

If the -R option is not present, the library search path that is recorded in the output object and passed to the runtime linker depends upon the target architecture instruction specified by the -xarch option (when -xarch is not present, -xarch=generic is assumed).

Examine the output from –dryrun and the –R option passed to the linker, \d, to see the default paths assumed by the compiler.

Interactions

This option accumulates instead of overrides.

If the LD_RUN_PATH environment variable is defined and the –R option is specified, then the path from –R is scanned and the path from LD_RUN_PATH is ignored.

See also

–norunpath, Linker and Libraries Guide
A.2.85  --readme
Same as -xhelp=readme.

A.2.86  --S
Compiles and generates only assembly code.
This option causes the CC driver to compile the program and output an assembly source file, without assembling the program. The assembly source file is named with a .s suffix.

A.2.87  --s
Strips the symbol table from the executable file.
This option removes all symbol information from output executable files. This option is passed to ld.

A.2.88  --sb
Obsolete and silently ignored.

A.2.89  --sbfast
Obsolete and silently ignored.

A.2.90  -staticlib=\[/,\ldots\]
Indicates which C++ libraries, specified by the -library option (including its defaults), by the -xlang option, and by the -xia option, are to be linked statically.

A.2.90.1 Values
/l must be one of the following values.
### TABLE A-20  The -staticlib Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]library</td>
<td>[Do not] link library statically. The valid values for library are all the valid values for -library (except %all and %none), all the valid values for -xlang, and interval (to be used in conjunction with -xia).</td>
</tr>
<tr>
<td>%all</td>
<td>Statically link all the libraries specified in the -library option, all the libraries specified in the -xlang option, and, if -xia is specified in the command line, the interval libraries.</td>
</tr>
<tr>
<td>%none</td>
<td>Link no libraries specified in the -library option and the -xlang option statically. If -xia is specified in the command line, link no interval libraries statically.</td>
</tr>
</tbody>
</table>

### Defaults

If -staticlib is not specified, -staticlib=%none is assumed.

### Examples

The following command line links libCrun statically because Crun is a default value for -library:

```
example% CC -staticlib=Crun (correct)
```

However, the following command line does not link libgc because libgc is not linked unless explicitly specified with the -library option:

```
example% CC -staticlib=gc (incorrect)
```

To link libgc statically, use the following command:

```
example% CC -library=gc -staticlib=gc (correct)
```

With the following command, the librwrite library is linked dynamically. Because librwrite is not a default library and is not selected using the -library option, -staticlib has no effect:

```
example% CC -lwbwrite -library=iostream \ 
-staticlib=rlwtools7 (incorrect)
```

This command links the librwrite library statically:

```
example% CC -library=rlwtools7,iostream -staticlib=rlwtools7 (correct)
```

This command will link the Sun Performance Libraries dynamically because -library=sunperf must be used in conjunction with -staticlib=sunperf in order for the -staticlib option to have an effect on the linking of these libraries:
This command links the Sun Performance Libraries statically:

```
example% CC -library=sunperf -staticlib=sunperf (correct)
```

**Interactions**

This option accumulates instead of overrides.

The `-staticlib` option only works for the C++ libraries that are selected explicitly with the `-xia` option, the `-xlang` option, and the `-library` option, in addition to the C++ libraries that are selected implicitly by default. In compatibility mode (`-compat=[4]`), `libC` is selected by default. In standard mode (the default mode), `Cstd` and `Crun` are selected by default.

When using `-xarch=v9`, `-xarch=v9a`, or `-xarch=v9b` (or equivalent 64-bit architecture options), some C++ libraries are not available as static libraries.

**Warnings**

The set of allowable values for `library` is not stable and might change from release to release.

When using `-xarch=v9`, `-xarch=v9a`, or `-xarch=v9b`, (or equivalent 64-bit architecture options), some libraries are not available as static libraries.

The options `-staticlib=Crun` and `-staticlib=Cstd` do not work on 64-bit Solaris x86 platforms. Sun recommends against linking these libraries statically on any platform. In some cases, static linking can prevent a program from working.

**See also**

- `-library`, "12.5 Statically Linking Standard Libraries" on page 147

### A.2.91 `-sync_stdio=[yes|no]`

Use this option when your run-time performance is degraded due to the synchronization between C++ iostreams and C stdio. Synchronization is needed only when you use iostreams to write to cout and stdio to write to stdout in the same program. The C++ standard requires synchronization so the C++ compiler turns it on by default. However, application performance is often much better without synchronization. If your program does not write to both cout and stdout, you can use the option `-sync_stdio=no` to turn off synchronization.

#### A.2.91.1 Defaults:

If you do not specify `-sync_stdio`, the compiler sets it to `-sync_stdio=yes`. 
Examples:

Consider the following example:

```c
#include <stdio.h>
#include <iostream>
int main()
{
    std::cout << "Hello ";
    printf("beautiful ");
    std::cout << "world!";
    printf("\n");
}
```

With synchronization, the program prints on a line by itself

Hello beautiful world!

Without synchronization, the output gets scrambled.

Warnings:

This option is only effective for linking of executables, not for libraries.

A.2.92 \(-\text{temp}=\text{path}\)

Defines the directory for temporary files.

This option sets the path name of the directory for storing the temporary files which are generated during the compilation process. The compiler gives precedence to the value set by \(-\text{temp}\) over the value of \(\text{TMPDIR}\).

A.2.92.1 See also

\(-\text{keeptmp}\)

A.2.93 \(-\text{template}=\text{opt}, \text{opt}...\)

Enables/disables various template options.

A.2.93.1 Values

\(\text{opt}\) must be one of the following values.
### TABLE A-21  The -template Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]extdef</td>
<td>[Do not] Search for template definitions in separate source files.</td>
</tr>
<tr>
<td>[no%]geninlinefuncs</td>
<td>[Do not] Generate unreferenced inline member functions for explicitly instantiated class templates.</td>
</tr>
<tr>
<td>[no%]wholeclass</td>
<td>[Do not] Instantiate a whole template class, rather than only those functions that are used. You must reference at least one member of the class; otherwise, the compiler does not instantiate any members for the class.</td>
</tr>
</tbody>
</table>

### Defaults

If the -template option is not specified, -template=no%wholeclass,extdef is assumed.

### Examples

Consider the following code:

```cpp
example% cat Example.cc
    template <class T> struct S {
    void imf() {}
    static void smf() {}
    };

    template class S <int>;

    int main() {
    }
example%
```

When you specify -template=geninlinefuncs, even though the two member functions of S are not called in the program, they are generated in the object file.

```bash
example% CC -c -template=geninlinefuncs Example.cc
eexample% nm -C Example.o
```

Example.o:

```
[Index] Value Size Type  Bind Other Shndx Name
[5]    0    0 NOTY  GLOB  0  ABS  __fsr_init_value
[1]    0    0 FILE  LOCL  0  ABS  b.c
[4]   16   32 FUNC  GLOB  0  2  main
[3]  104   24 FUNC  LOCL  0  2  void S<int>::imf()
    [__lpBS4Ci_Dimf6M_v_]
```
A.2.94  
**–time**

Same as `–xtime`.

A.2.95  
**–Uname**

Deletes initial definition of the preprocessor symbol `name`.

This option removes any initial definition of the macro symbol `name` created by `-D` on the command line including those implicitly placed there by the CC driver. This option has no effect on any other predefined macros, nor on macro definitions in source files.

To see the `-D` options that are placed on the command line by the CC driver, add the `-dryrun` option to your command line.

A.2.95.1  
**Examples**

The following command undefines the predefined symbol `__sun`. Preprocessor statements in `foo.cc` such as `#ifdef(__sun)` will sense that the symbol is undefined.

```
example% CC -U__sun foo.cc
```

**Interactions**

You can specify multiple `-U` options on the command line.

All `-U` options are processed after any `-D` options that are present. That is, if the same `name` is specified for both `-D` and `-U` on the command line, `name` is undefined, regardless of the order the options appear.

**See also**

`-D`

A.2.96  
**–unroll=n**

Same as `–xunroll=n`. 
A.2.97  

–V

Same as –verbose=version.

A.2.98  

–v

Same as –verbose=diags.

A.2.99  

–vdelx

Deprecated, do not use.

Compatibility mode only (–compat [=4]):

For expressions using delete[], this option generates a call to the runtime library function _vector_deletex_ instead of generating a call to _vector_delete_. The function _vector_delete_ takes two arguments: the pointer to be deleted and the size of each array element.

The function _vector_deletex_ behaves the same as _vector_delete_ except that it takes a third argument: the address of the destructor for the class. This third argument is not used by the function, but is provided to be used by third-party vendors.

A.2.99.1 Default

The compiler generates a call to _vector_delete_ for expressions using delete[].

Warnings

This is an obsolete option that will be removed in future releases. Do not use this option unless you have bought some software from a third-party vendor and the vendor recommends using this option.

A.2.100  

–verbose=v[, v...

Controls compiler verbosity.

A.2.100.1 Values

v must be one of the following values.
### TABLE A-22 The -verbose Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]diags</td>
<td>[Do not] Print the command line for each compilation pass.</td>
</tr>
<tr>
<td>[no%]template</td>
<td>[Do not] Turn on the template instantiation verbose mode (sometimes called the “verify” mode). The verbose mode displays each phase of instantiation as it occurs during compilation.</td>
</tr>
<tr>
<td>[no%]version</td>
<td>[Do not] Direct the CC driver to print the names and version numbers of the programs it invokes.</td>
</tr>
<tr>
<td>%all</td>
<td>Invokes all of the above.</td>
</tr>
<tr>
<td>%none</td>
<td>-verbose=%none is the same as -verbose=no%template,no%diags,no%version.</td>
</tr>
</tbody>
</table>

### Defaults

If -verbose is not specified, -verbose=%none is assumed.

### Interactions

This option accumulates instead of overrides.

#### A.2.101 +w

Identifies code that might have unintended consequences. The +w option no longer generates a warning if a function is too large to inline or if a declared program element is unused. These warnings do not identify real problems in the source, and were thus inappropriate to some development environments. Removing these warnings from +w enables more aggressive use of +w in those environments. These warnings are still available with the +w2 option.

This option generates additional warnings about questionable constructs that are:

- Nonportable
- Likely to be mistakes
- Inefficient

#### A.2.101.1 Defaults

If +w is not specified, the compiler warns about constructs that are almost certainly problems.

### See also

- w, +w2
A.2.102  
+w2

Emits all the warnings emitted by +w plus warnings about technical violations that are probably harmless, but that might reduce the maximum portability of your program.

The +w2 option no longer warns about the use of implementation-dependent constructs in the system header files. Because the system header files are the implementation, the warning was inappropriate. Removing these warnings from +w2 enables more aggressive use of the option.

A.2.102.1  See also
+w

A.2.103  
-w

Suppresses most warning messages.

This option causes the compiler not to print warning messages. However, some warnings, particularly warnings regarding serious anachronisms, cannot be suppressed.

A.2.103.1  See also
+w

A.2.104  
-Xm

Same as -features=iddollar.

A.2.105  
-xa

Generates code for profiling.

If set at compile time, the TC0VDIR environment variable specifies the directory where the coverage (.d) files are located. If this variable is not set, then the coverage (.d) files remain in the same directory as the source files.

Use this option only for backward compatibility with old coverage files.

A.2.105.1  Interactions

The -xprofile=tcov option and the -xa option are compatible in a single executable. That is, you can link a program that contains some files that have been compiled with -xprofile=tcov, and others that have been compiled with -xa. You cannot compile a single file with both options.
The –xa option is incompatible with –g.

Warnings
If you compile and link in separate steps and you compile with -xa, be sure to link with –xa, or you might get unexpected results.

See also
–xprofile=tcov, tcov(1) man page, Program Performance Analysis Tools.

A.2.106 -xaddr32
(Solaris x86/x64 only) The -xaddr32=yes compilation flag restricts the resulting executable or shared object to a 32-bit address space.

An executable that is compiled in this manner results in the creation of a process that is restricted to a 32-bit address space.

When -xaddr32=no is specified a usual 64 bit binary is produced.

If the -xaddr32 option is not specified, -xaddr32=no is assumed.

If only -xaddr32 is specified -xaddr32=yes is assumed.

This option is only applicable to -m64 compilations and only on Solaris platforms supporting SF1_SUNW_ADDR32 software capability. Since Linux kernels do not support address space limitation, this option is not available on Linux.

When linking, if a single object file was compiled with -xaddr32=yes the whole output file is assumed to be compiled with -xaddr32=yes.

A shared object that is restricted to a 32-bit address space must be loaded by a process that executes within a restricted 32-bit mode address space.

For more information refer to the SF1_SUNW_ADDR32 software capabilities definition, described in the Linker and Libraries Guide.

A.2.107 -xalias_level[=n]
(SPARC) The C++ compiler can perform type-based alias-analysis and optimizations when you specify the following command:

- -xalias_level[=n]
  where n is any, simple, or compatible.
- `xalias_level=any`

  At this level of analysis, the compiler assumes that any type may alias any other type. However, despite this assumption, some optimization is possible.

- `xalias_level=simple`

  The compiler assumes that simple types are not aliased. Specifically, a storage object with a dynamic type that is one of the following simple types:

<table>
<thead>
<tr>
<th>char</th>
<th>short int</th>
<th>long int</th>
<th>float</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed char</td>
<td>unsigned short int</td>
<td>unsigned long int</td>
<td>double</td>
</tr>
<tr>
<td>unsigned char</td>
<td>int</td>
<td>long long int</td>
<td>long double</td>
</tr>
<tr>
<td>wchar_t</td>
<td>unsigned int</td>
<td>unsigned long long int</td>
<td>enumeration types</td>
</tr>
<tr>
<td>data pointer types</td>
<td>function pointer types</td>
<td>data member pointer types</td>
<td>function member pointer types</td>
</tr>
</tbody>
</table>

is only accessed through lvalues of the following types:

- The dynamic type of the object
  - A constant or volatile qualified version of the dynamic type of the object, a type that is the signed or unsigned type corresponding to the dynamic type of the object
  - A type that is the signed or unsigned type corresponding to a constant or volatile qualified version of the dynamic type of the object
  - An aggregate or union type that includes one of the aforementioned types among its members (including, recursively, a member of a subaggregate or contained union)
  - A char or unsigned char type

- `xalias_level=compatible`

  The compiler assumes that layout-incompatible types are not aliased. A storage object is only accessed through lvalues of the following types:

- The dynamic type of the object
  - A constant or volatile qualified version of the dynamic type of the object, a type that is the signed or unsigned type which corresponds to the dynamic type of the object
  - A type that is the signed or unsigned type which corresponds to the constant or volatile qualified version of the dynamic type of the object
  - An aggregate or union type that includes one of the aforementioned types among its members (including, recursively, a member of a subaggregate or contained union)
  - A type that is (possibly constant or volatile qualified) base class type of the dynamic type of the object
A char or unsigned char type.
The compiler assumes that the types of all references are layout compatible with the
dynamic type of the corresponding storage object. Two types are layout-compatible
under the following conditions:

- If two types are the same type, then they are layout-compatible types.
- If two types differ only in constant or volatile qualification, then they are
  layout-compatible types.
- For each of the signed integer types, there exists a corresponding (but different)
  unsigned integer type. These corresponding types are layout compatible.
- Two enumeration types are layout-compatible if they have the same underlying type.
- Two Plain Old Data (POD) struct types are layout compatible if they have the same
  number of members, and corresponding members (in order) have layout compatible
  types.
- Two POD union types are layout compatible if they have the same number of members,
  and corresponding members (in any order) have layout compatible types.

References may be non-layout-compatible with the dynamic type of the storage object
under limited circumstances:

- If a POD union contains two or more POD structs that share a common initial sequence,
  and if the POD union object currently contains one of those POD structs, it is permitted
  to inspect the common initial part of any of them. Two POD structs share a common
  initial sequence if corresponding members have layout compatible types and, as
  applicable to bit fields, the same widths, for a sequence of one or more initial members.
- A pointer to a POD struct object, suitably converted using a reinterpret_cast, points to
  its initial member, or if that member is a bit field, to the unit in which it resides.

### A.2.107.1 Defaults

If you do not specify -xalias_level, the compiler sets the option to -xalias_level=any. If
you specify -xalias_level but do not provide a value, the compiler sets the option to
-xalias_level=compatible.

### Interactions

The compiler does not perform type-based alias analysis at optimization level -xO2 and below.

### Warning

If you are using reinterpret_cast or an equivalent old-style cast, the program may violate the
assumptions of the analysis. Also, union type punning, as shown in the following example,
violates the assumptions of the analysis.
union bitbucket{
    int i;
    float f;
};

int bitsof(float f){
    bitbucket var;
    var.f=3.6;
    return var.i;
}

A.2.108 -xannotate[=yes|no]

(Solaris) Instructs the compiler to create binaries that can later be transformed by binary
modification tools like binopt(1). Future binary analysis, code coverage and memory error
detection tools will also work with binaries built with this option.

Use the -xannotate=no option to prevent the modification of the binary file by these tools.

The -xannotate=yes option must be used with optimization level -xO1 or higher to be effective,
and it is only effective on systems with the new linker support library interface -ld_open(). If
the compiler is used on an operating system without this linker interface, such as the Solaris 9
OS, the compiler silently reverts to -xannotate=no. The new linker interface is available in
Solaris 10 patch 127111-07, Solaris 10 Update 5 and OpenSolaris.

The default is -xannotate=yes, but if either of the above conditions is not met, the default
reverts to -xannotate=no.

This option is not available on Linux platforms.

A.2.109 -xar

Creates archive libraries.

When building a C++ archive that uses templates, it is necessary to include in the archive those
template functions that are instantiated in the template repository. The template repository is
used only when at least one object file was compiled using the -instances=extern option.
Using this option automatically adds those templates to the archive as needed.

A.2.109.1 Values

Specify -xar to invokes ar -c -r and create an archive from scratch.
Examples

The following command line archives the template functions contained in the library and object files.

```
example% CC -xar -o libmain.a a.o b.o c.o
```

Warnings

Do not add .o files from the template database on the command line.

Do not use the ar command directly for building archives. Use CC –xar to ensure that template instantiations are automatically included in the archive.

See also

ar(1), Table 15–3

A.2.110

−xarch=isa

Specifies the target instruction set architecture (ISA).

This option limits the code generated by the compiler to the instructions of the specified instruction set architecture. This option does not guarantee use of any target–specific instructions. However, use of this option may affect the portability of a binary program.

**Note** – Use the -m64 or -m32 option to specify the intended memory model, LP64 (64-bits) or ILP32 (32-bits) respectively. The -xarch option no longer indicates the memory model, except for compatibility with previous releases, as indicated below.

If you compile and link in separate steps, make sure you specify the same value for -xarch in both steps. For complete list of all compiler options that must be specified at both compile time and at link time, see “3.3.3 Compile-Time and Link-Time Options” on page 50.

A.2.110.1

- xarch Flags for SPARC

The following table gives the details for each of the -xarch keywords on SPARC platforms.
<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>generic</strong></td>
<td>Uses the instruction set common to most processors. This is the default, and is equivalent to <code>v8plus</code> when compiling with <code>-m32</code>, and <code>sparc</code> with <code>-m64</code>.</td>
</tr>
<tr>
<td><strong>generic64</strong></td>
<td>Compile for good performance on most 64-bit platforms. (Solaris only). This option is equivalent to <code>-m64 -xarch=generic</code> and is provided for compatibility with earlier releases. Use <code>-m64</code> to specify 64-bit compilation instead of <code>-xarch=generic64</code>.</td>
</tr>
<tr>
<td><strong>native</strong></td>
<td>Compile for good performance on this system. The compiler chooses the appropriate setting for the current system processor it is running on.</td>
</tr>
<tr>
<td><strong>native64</strong></td>
<td>Compile for good performance on this system (Solaris only). This option is equivalent to <code>-m64 -xarch=native</code> and is provided for compatibility with earlier releases.</td>
</tr>
<tr>
<td><strong>sparc</strong></td>
<td>Compile for the SPARC-V9 ISA, but without the Visual Instruction Set (VIS), and without other implementation-specific ISA extensions. This option enables the compiler to generate code for good performance on the V9 ISA.</td>
</tr>
<tr>
<td><strong>sparcvis</strong></td>
<td>Compile for SPARC-V9 plus the Visual Instruction Set (VIS) version 1.0, and with UltraSPARC extensions. This option enables the compiler to generate code for good performance on the UltraSPARC architecture.</td>
</tr>
<tr>
<td><strong>sparcvis2</strong></td>
<td>Enables the compiler to generate object code for the UltraSPARC architecture, plus the Visual Instruction Set (VIS) version 2.0, and with UltraSPARC III extensions.</td>
</tr>
<tr>
<td><strong>sparcfmaf</strong></td>
<td>Enables the compiler to use instructions from the SPARC-V9 instruction set, plus the UltraSPARC extensions, including the Visual Instruction Set (VIS) version 1.0, the UltraSPARC-III extensions, including the Visual Instruction Set (VIS) version 2.0, and the SPARC64 VI extensions for floating-point multiply-add. You must use <code>-xarch=sparcfmaf</code> in conjunction with <code>fma=fused</code> and some optimization level to get the compiler to attempt to find opportunities to use the multiply-add instructions automatically.</td>
</tr>
<tr>
<td><strong>v7</strong></td>
<td>(Obsolete) Compile for the SPARC-V7 ISA. Current Solaris operating systems no longer support the SPARC V7 architecture, and programs compiled with this option run slower on current platforms. The default is <code>-xarch=v8plus</code>. Examples: SPARCstation 1, SPARCstation 2.</td>
</tr>
<tr>
<td>Flag</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>v8a</td>
<td>(Obsolete) Compile for the V8a version of the SPARC-V8 ISA. By definition, V8a means the V8 ISA, but without the <code>fsmul</code>d instruction. This option enables the compiler to generate code for good performance on the V8a ISA. Example: Any system based on the microSPARC I chip architecture</td>
</tr>
<tr>
<td>v8</td>
<td>(Obsolete) Compile for the SPARC-V8 ISA. Enables the compiler to generate code for good performance on the V8 architecture. Example: SPARCstation 10</td>
</tr>
<tr>
<td>v8plus</td>
<td>Compile for the V8plus version of the SPARC-V9 ISA. By definition, V8plus means the V9 ISA, but limited to the 32-bit subset defined by the V8plus ISA specification, without the Visual Instruction Set (VIS), and without other implementation-specific ISA extensions. ■ This option enables the compiler to generate code for good performance on the V8plus ISA. ■ The resulting object code is in SPARC-V8+ ELF32 format and only executes in a Solaris UltraSPARC environment—it does not run on a V7 or V8 processor. Example: Any system based on the UltraSPARC chip architecture</td>
</tr>
<tr>
<td>v8plusa</td>
<td>Compile for the V8plusa version of the SPARC-V9 ISA. By definition, V8plusa means the V8plus architecture, plus the Visual Instruction Set (VIS) version 1.0, and with UltraSPARC extensions. ■ This option enables the compiler to generate code for good performance on the UltraSPARC architecture, but limited to the 32-bit subset defined by the V8plus specification. ■ The resulting object code is in SPARC-V8+ ELF32 format and only executes in a Solaris UltraSPARC environment—it does not run on a V8 processor. Example: Any system based on the UltraSPARC chip architecture</td>
</tr>
<tr>
<td>v8plusb</td>
<td>Compile for the V8plusb version of the SPARC-V8plus ISA with UltraSPARC III extensions. Enables the compiler to generate object code for the UltraSPARC architecture, plus the Visual Instruction Set (VIS) version 2.0, and with UltraSPARC III extensions. ■ The resulting object code is in SPARC-V8+ ELF32 format and executes only in a Solaris UltraSPARC III environment. ■ Compiling with this option uses the best instruction set for good performance on the UltraSPARC III architecture.</td>
</tr>
</tbody>
</table>
### The -xarch Flags for SPARC Platforms (Continued)

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>v9</td>
<td>Is equivalent to -m64 -xarch=sparc. Legacy makefiles and scripts that use -xarch=v9 to obtain the 64-bit memory model need only use -m64.</td>
</tr>
<tr>
<td>v9a</td>
<td>Is equivalent to -m64 -xarch=sparcvis and is provided for compatibility with earlier releases.</td>
</tr>
<tr>
<td>v9b</td>
<td>Is equivalent to -m64 -xarch=sparcvis2 and is provided for compatibility with earlier releases.</td>
</tr>
</tbody>
</table>

Also note the following:

- SPARC instruction set architectures V7, V8 and V8a are obsolete and should not be used.
- Object binary files (.o) compiled with v8plus and v8plusa can be linked and can execute together, but only on a SPARC V8plusa compatible platform.
- Object binary files (.o) compiled with v8plus, v8plusa, and v8plusb can be linked and can execute together, but only on a SPARC V8plusb compatible platform.
- -xarch values v9, v9a, and v9b are only available on UltraSPARC 64-bit Solaris operating systems.
- Object binary files (.o) compiled with generic64, native64, v9 and v9a can be linked and can execute together, but will run only on a SPARC V9a compatible platform.
- Object binary files (.o) compiled with generic64, native64, v9, v9a, and v9b can be linked and can execute together, but will run only on a SPARC V9b compatible platform.

For any particular choice, the generated executable may run much more slowly on earlier architectures. Also, although quad-precision (long double) floating-point instructions are available in many of these instruction set architectures, the compiler does not use these instructions in the code it generates.

### A.2.110.2 -xarch Flags for x86

The following table lists the -xarch flags on x86 platforms.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>amd64</td>
<td>Is equivalent to -m64 -xarch=sse2 (Solaris only). Legacy makefiles and scripts that use -xarch=amd64 to obtain the 64-bit memory model need only use -m64.</td>
</tr>
<tr>
<td>amd64a</td>
<td>Is equivalent to -m64 -xarch=sse2a (Solaris only).</td>
</tr>
<tr>
<td>generic</td>
<td>Uses the instruction set common to most processors. This is the default, and is equivalent to pentium_pro when compiling with -m32, and sse2 with -m64.</td>
</tr>
<tr>
<td>Flag</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>generic64</td>
<td>Compile for good performance on most 64-bit platforms. (Solaris only). This option is equivalent to -m64 -xarch=generic and is provided for compatibility with earlier releases. Use -m64 to specify 64-bit compilation instead of \texttt{-xarch=generic64}.</td>
</tr>
<tr>
<td>native</td>
<td>Compile for good performance on this system. The compiler chooses the appropriate setting for the current system processor it is running on.</td>
</tr>
<tr>
<td>native64</td>
<td>Compile for good performance on this system (Solaris only). This option is equivalent to -m64 -xarch=native and is provided for compatibility with earlier releases.</td>
</tr>
<tr>
<td>pentium_pro</td>
<td>Limits the instruction set to the 32-bit pentium_pro architecture.</td>
</tr>
<tr>
<td>pentium_proa</td>
<td>Adds the AMD extensions (3DNow!, 3DNow! extensions, and MMX extensions) to the 32-bit pentium_pro architecture.</td>
</tr>
<tr>
<td>sse</td>
<td>Adds the SSE instruction set to the pentium_pro architecture.</td>
</tr>
<tr>
<td>ssea</td>
<td>Adds the AMD extensions (3DNow!, 3DNow! extensions, and MMX extensions) to the 32-bit SSE architecture.</td>
</tr>
<tr>
<td>sse2</td>
<td>Adds the SSE2 instruction set to the pentium_pro architecture.</td>
</tr>
<tr>
<td>sse2a</td>
<td>Adds the AMD extensions (3DNow!, 3DNow! extensions, and MMX extensions) to the 32-bit SSE2 architecture.</td>
</tr>
<tr>
<td>sse3</td>
<td>Adds the SSE3 instruction set to SSE2 instruction set.</td>
</tr>
<tr>
<td>ssse3</td>
<td>Supplements the pentium_pro, SSE, SSE2, and SSE3 instruction sets with the SSE3 instruction set.</td>
</tr>
<tr>
<td>sse4_1</td>
<td>Supplements the pentium_pro, SSE, SSE2, SSE3, and SSSE3 instruction sets with the SSE4.1 instruction set.</td>
</tr>
<tr>
<td>sse4_2</td>
<td>Supplements the pentium_pro, SSE, SSE2, SSE3, SSSE3, and SSE4.1 instruction sets with the SSE4.2 instruction set.</td>
</tr>
</tbody>
</table>

### A.2.110.3 Special x86 Notes

There are some important issues to be aware of when compiling for x86 Solaris platforms.

The legacy Sun-style parallelization pragmas are not available on x86. Use OpenMP instead. See the *Sun Studio OpenMP API User’s Guide* for information on converting legacy parallelization directives to OpenMP.

Programs compiled with -xarch set to sse, sse2, sse2a, sse3, or beyond must be run only on platforms that provide these extensions and features.
Solaris OS releases are SSE/SSE2-enabled on Pentium 4-compatible platforms. Earlier versions of Solaris OS are not SSE/SSE2-enabled. If an instruction set selected by \texttt{-xarch} is not enabled in the running Solaris OS, the compiler will not be able to generate or link code for that instruction set.

If you compile and link in separate steps, always link using the compiler and using the same \texttt{-xarch} setting to ensure that the correct startup routine is linked.

Numerical results on x86 may differ from results on SPARC due to the x86 80-bit floating-point registers. To minimize these differences, use the \texttt{-fstore} option or compile with \texttt{-xarch=sse2} if the hardware supports SSE2.

Numerical results can also differ between Solaris and Linux because the intrinsic math libraries (for example, \texttt{sin(x)}) are not the same.

\section*{A.2.110.4 Binary Compatibility Verification}

Starting with Sun Studio 11 and the Solaris 10 OS, program binaries compiled and built using these specialized \texttt{-xarch} hardware flags are verified that they are being run on the appropriate platform.

On systems prior to Solaris 10, no verification is done and it is the user’s responsibility to ensure objects built using these flags are deployed on suitable hardware.

Running programs compiled with these \texttt{-xarch} options on platforms that are not enabled with the appropriate features or instruction set extensions could result in segmentation faults or incorrect results occurring without any explicit warning messages.

This warning extends also to programs that employ \texttt{.il} inline assembly language functions or \texttt{__asm()} assembler code that utilize SSE, SSE2, SSE2a, and SSE3 instructions and extensions.

\section*{A.2.110.5 Interactions}

Although this option can be used alone, it is part of the expansion of the \texttt{-xtarget} option and may be used to override the \texttt{-xarch} value that is set by a specific \texttt{-xtarget} option. For example, \texttt{-xtarget=ultra2} expands to \texttt{-xarch=v8plusa -xchip=ultra2 -xcache=16/32/1:512/64/1}. In the following command \texttt{-xarch=v8plusb} overrides the \texttt{-xarch=v8plusa} that is set by the expansion of \texttt{-xtarget=ultra2}.

\begin{verbatim}
example% CC -xtarget=ultra2 -xarch=v8plusb foo.cc
\end{verbatim}

Use of \texttt{-compat[=4]} with \texttt{-xarch=generic64}, \texttt{-xarch=native64}, \texttt{-xarch=v9}, \texttt{-xarch=v9a}, or \texttt{-xarch=v9b} is not supported.
A.2.110.6 **Warnings**

If you use this option with optimization, the appropriate choice can provide good performance of the executable on the specified architecture. An inappropriate choice, however, might result in serious degradation of performance or in a binary program that is not executable on the intended target platform.

If you compile and link in separate steps, make sure you specify the same value for \texttt{-xarch} in both steps.

A.2.111 **-xautopar**

\textbf{Note} – This option does not accept OpenMP parallelization directives. The Sun-specific MP pragmas have been deprecated and are no longer supported. See the \textit{Sun Studio OpenMP API User's Guide} for migration information to the directives of the standard.

(SPARC) Turns on automatic parallelization for multiple processors. Does dependence analysis (analyze loops for inter-iteration data dependence) and loop restructuring. If optimization is not at \texttt{-xO3} or higher, optimization is raised to \texttt{-xO3} and a warning is issued.

Avoid \texttt{-xautopar} if you do your own thread management.

To achieve faster execution, this option requires a multiple processor system. On a single-processor system, the resulting binary usually runs slower.

To run a parallelized program in a multithreaded environment, you must set the \texttt{OMP_NUM_THREADS} environment variable prior to execution. See the \textit{Sun Studio OpenMP API User's Guide} for more information.

If you use \texttt{-xautopar} and compile and link in one step, then linking automatically includes the microtasking library and the threads-safe C runtime library. If you use \texttt{-xautopar} and compile and link in separate steps, then you must also link with \texttt{-xautopar}.

A.2.111.1 **See Also**

"A.2.158 -xopenmp\[=i\]" on page 324

A.2.112 **-xbinopt={prepare|off}\)**

(SPARC) Instructs the compiler to prepare the binary for later optimizations, transformations and analysis, see \texttt{binopt(1)}. This option may be used for building executables or shared objects. If you compile in separate steps, \texttt{-xbinopt} must appear on both compile and link steps:
example% cc -c -xO1 -xbinopt=prepare a.c b.c
example% cc -o myprog -xbinopt=prepare a.o

If some source code is not available for compilation, this option may still be used to compile the remainder of the code. It should then be used in the link step that creates the final binary. In such a situation, only the code compiled with this option can be optimized, transformed or analyzed.

A.2.112.1 Defaults
The default is -xbinopt=off.

Interactions
This option must be used with optimization level -xO1 or higher to be effective. There is a modest increase in size of the binary when built with this option.

Compiling with -xbinopt=prepare and -g increases the size of the executable by including debugging information.

A.2.113 -xbuiltin[=%all|%none]]
Enables or disables better optimization of standard library calls.

By default, the functions declared in standard library headers are treated as ordinary functions by the compiler. However, some of those functions can be recognized as “intrinsic” or “built-in” by the compiler. When treated as a built-in, the compiler can generate more efficient code. For example, the compiler can recognize that some functions have no side effects, and always return the same output given the same input. Some functions can be generated inline directly by the compiler. See the er_src(1) man page for an explanation of how to read compiler commentary in object files to determine for which functions the compiler actually makes a substitution.

The -xbuiltin=%all option asks the compiler to recognize as many of the built-in standard functions as possible. The exact list of recognized functions varies with the version of the compiler code generator.

The -xbuiltin=%none option results in the default compiler behavior, and the compiler does not do any special optimizations for built-in functions.

A.2.113.1 Defaults
If the -xbuiltin option is not specified, then the compiler assumes -xbuiltin=%none.

If only -xbuiltin is specified, then the compiler assumes -xbuiltin=%all.

Interactions
The expansion of the macro -fast includes -xbuiltin=%all.
Examples
The following compiler command requests special handling of the standard library calls.

example% CC -xbuiltin -c foo.cc

The following compiler command request that there be no special handling of the standard library calls. Note that the expansion of the macro -fast includes -xbuiltin=%all.

example% CC -fast -xbuiltin=%none -c foo.cc

A.2.114 –xcache=c
Defines cache properties for use by the optimizer. This option does not guarantee that any particular cache property is used.

Note – Although this option can be used alone, it is part of the expansion of the -xtarget option; its primary use is to override a value supplied by the -xtarget option.

This release introduces an optional property [/ti] which sets the number of threads that can share the cache.

A.2.114.1 Values
c must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>generic</td>
<td>This is the default value which directs the compiler to use cache properties for good performance on most x86 and SPARC processors, without major performance degradation on any of them. With each new release, these best timing properties will be adjusted, if appropriate.</td>
</tr>
<tr>
<td>native</td>
<td>Set the parameters for the best performance on the host environment.</td>
</tr>
<tr>
<td>$i/l_i/a_i[/t_i]$</td>
<td>Defines level 1 cache properties</td>
</tr>
<tr>
<td>$s_i/l_i/a_i[/t_i]$</td>
<td>Defines level 1 and 2 cache properties</td>
</tr>
<tr>
<td>$s_i/l_i/a_i[/t_i]$</td>
<td>Defines level 1, 2, and 3 cache properties</td>
</tr>
</tbody>
</table>

The definitions of the cache properties $s_i/l_i/a_i[/t_i]$ are as follows:
The size of the data cache at level \( i \), in kilobytes

The line size of the data cache at level \( i \), in bytes

The associativity of the data cache at level \( i \)

For example, \( i = 1 \) designates level 1 cache properties, \( s_1/l_1/a_1 \).

**Defaults**

If \(-\text{xcache}\) is not specified, the default \(-\text{xcache=generic}\) is assumed. This value directs the compiler to use cache properties for good performance on most SPARC processors, without major performance degradation on any of them.

If you do not specify a value for \( t \), the default is 1.

**Examples**

\(-\text{xcache}=16/32/4:1024/32/1\) specifies the following:

<table>
<thead>
<tr>
<th>Level 1 Cache Has</th>
<th>Level 2 Cache Has</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 Kbytes</td>
<td>1024 Kbytes</td>
</tr>
<tr>
<td>32 bytes line size</td>
<td>32 bytes line size</td>
</tr>
<tr>
<td>4-way associativity</td>
<td>Direct mapping associativity</td>
</tr>
</tbody>
</table>

**See also**

\(-\text{xtarget}=t\)

**A.2.115 -xcg[89|92]**

(SPARC) Obsolete, do not use this option. Current Solaris operating system software no longer support SPARC V7 architecture. Compiling with this option generates code that runs slower on current SPARC platforms. Use -0 instead and take advantage of compiler defaults for -xarch, -xchip, and -xcache.
A.2.116  -xchar[=o]

The option is provided solely for the purpose of easing the migration of code from systems where the char type is defined as unsigned. Unless you are migrating from such a system, do not use this option. Only code that relies on the sign of a char type needs to be rewritten to explicitly specify signed or unsigned.

A.2.116.1  Values

You can substitute one of the following for o:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed</td>
<td>Treat character constants and variables declared as char as signed. This impacts the behavior of compiled code, it does not affect the behavior of library routines.</td>
</tr>
<tr>
<td>s</td>
<td>Equivalent to signed</td>
</tr>
<tr>
<td>unsigned</td>
<td>Treat character constants and variables declared as char as unsigned. This impacts the behavior of compiled code, it does not affect the behavior of library routines.</td>
</tr>
<tr>
<td>u</td>
<td>Equivalent to unsigned</td>
</tr>
</tbody>
</table>

Defaults

If you do not specify -xchar, the compiler assumes -xchar=s.

If you specify -xchar, but do not specify a value, the compiler assumes -xchar=s.

Interactions

The -xchar option changes the range of values for the type char only for code compiled with -xchar. This option does not change the range of values for type char in any system routine or header file. In particular, the value of CHAR_MAX and CHAR_MIN, as defined by limits.h, do not change when this option is specified. Therefore, CHAR_MAX and CHAR_MIN no longer represent the range of values encodable in a plain char.

Warnings

If you use -xchar=unsigned, be particularly careful when you compare a char against a predefined system macro because the value in the macro may be signed. This is most common for any routine that returns an error code which is accessed through a macro. Error codes are typically negative values so when you compare a char against the value from such a macro, the result is always false. A negative number can never be equal to any value of an unsigned type.
It is strongly recommended that you never use \texttt{-xchar} to compile routines for any interface exported through a library. The Solaris ABI specifies type \texttt{char} as signed, and system libraries behave accordingly. The effect of making \texttt{char} unsigned has not been extensively tested with system libraries. Instead of using this option, modify your code so that it does not depend on whether type \texttt{char} is signed or unsigned. The signedness of type \texttt{char} varies among compilers and operating systems.

\subsection{A.2.117 \texttt{-xcheck[=i]}}

SPARC: Compiling with \texttt{-xcheck=}\texttt{stkovf} adds a runtime check for stack overflow of the main thread in a singly-threaded program as well as slave-thread stacks in a multithreaded program. If a stack overflow is detected, a \texttt{SIGSEGV} is generated. If your application needs to handle a \texttt{SIGSEGV} caused by a stack overflow differently than it handles other address-space violations, see \texttt{sigaltstack}(2).

\subsubsection{A.2.117.1 Values}

\texttt{i} must be one of the following:

\begin{table}[h]
\centering
\begin{tabular}{ll}
\hline
Value & Meaning \\
\hline
\%all & Perform all checks. \\
\%none & Perform no checks. \\
\texttt{stkovf} & Turns on stack-overflow checking. \\
\texttt{no\_stkovf} & Turns off stack-overflow checking. \\
\hline
\end{tabular}
\caption{The \texttt{-xcheck} Values}
\end{table}

\textbf{Defaults}

If you do not specify \texttt{-xcheck}, the compiler defaults to \texttt{-xcheck=}\texttt{\%none}.

If you specify \texttt{-xcheck} without any arguments, the compiler defaults to \texttt{-xcheck=}\texttt{\%none}.

The \texttt{-xcheck} option does not accumulate on the command line. The compiler sets the flag in accordance with the last occurrence of the command.

\subsection{A.2.118 \texttt{-xchip=c}}

Specifies target processor for use by the optimizer.

The \texttt{-xchip} option specifies timing properties by specifying the target processor. This option affects:
The ordering of instructions—that is, scheduling
The way the compiler uses branches
The instructions to use in cases where semantically equivalent alternatives are available

**Note** – Although this option can be used alone, it is part of the expansion of the `-xtarget` option; its primary use is to override a value supplied by the `-xtarget` option.

### A.2.118.1 Values

`c` must be one of the following values.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Value</th>
<th>Optimize Using Timing Properties for</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARC</td>
<td>generic</td>
<td>good performance on most SPARC processors</td>
</tr>
<tr>
<td></td>
<td>native</td>
<td>good performance on the system on which the compiler is running</td>
</tr>
<tr>
<td></td>
<td>old</td>
<td>processors earlier than the SuperSPARC processor</td>
</tr>
<tr>
<td></td>
<td>sparc64vi</td>
<td>SPARC64 VI processor</td>
</tr>
<tr>
<td></td>
<td>sparc64vii</td>
<td>SPARC64 VII processor</td>
</tr>
<tr>
<td></td>
<td>super</td>
<td>SuperSPARC processor</td>
</tr>
<tr>
<td></td>
<td>super2</td>
<td>SuperSPARC II processor</td>
</tr>
<tr>
<td></td>
<td>micro</td>
<td>microSPARC processor</td>
</tr>
<tr>
<td></td>
<td>micro2</td>
<td>microSPARC II processor</td>
</tr>
<tr>
<td></td>
<td>hyper</td>
<td>hyperSPARC processor</td>
</tr>
<tr>
<td></td>
<td>hyper2</td>
<td>hyperSPARC II processor</td>
</tr>
<tr>
<td></td>
<td>powerup</td>
<td>the Weitek PowerUp processor</td>
</tr>
<tr>
<td></td>
<td>ultra</td>
<td>UltraSPARC processor</td>
</tr>
<tr>
<td></td>
<td>ultra2</td>
<td>UltraSPARC II processor</td>
</tr>
<tr>
<td></td>
<td>ultra2e</td>
<td>UltraSPARC IIe processor</td>
</tr>
<tr>
<td></td>
<td>ultra2i</td>
<td>UltraSPARC IIi processor</td>
</tr>
<tr>
<td></td>
<td>ultra3</td>
<td>UltraSPARC III processor</td>
</tr>
</tbody>
</table>
### TABLE A–28 The -xchip Values (Continued)

<table>
<thead>
<tr>
<th>Platform</th>
<th>Value</th>
<th>Optimize Using Timing Properties for</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ultra3cu</td>
<td>UltraSPARC III Cu processor</td>
</tr>
<tr>
<td></td>
<td>ultra3i</td>
<td>UltraSparc IIIi processors.</td>
</tr>
<tr>
<td></td>
<td>ultra4</td>
<td>UltraSPARC IV processors.</td>
</tr>
<tr>
<td></td>
<td>ultra4plus</td>
<td>UltraSPARC IVplus processor.</td>
</tr>
<tr>
<td></td>
<td>ultraT1</td>
<td>UltraSPARC T1 processor.</td>
</tr>
<tr>
<td></td>
<td>ultraT2</td>
<td>UltraSPARC T2 processor.</td>
</tr>
<tr>
<td></td>
<td>ultraT2plus</td>
<td>UltraSPARC T2+ processor.</td>
</tr>
<tr>
<td>x86</td>
<td>generic</td>
<td>most x86 processors</td>
</tr>
<tr>
<td></td>
<td>core2</td>
<td>Intel Core2 processor</td>
</tr>
<tr>
<td></td>
<td>nehalem</td>
<td>Intel Nehalem processor</td>
</tr>
<tr>
<td></td>
<td>opteron</td>
<td>AMD Opteron processor</td>
</tr>
<tr>
<td></td>
<td>penryn</td>
<td>Intel Penryn processor</td>
</tr>
<tr>
<td></td>
<td>pentium</td>
<td>Intel Pentium processor</td>
</tr>
<tr>
<td></td>
<td>pentium_pro</td>
<td>Intel Pentium Pro processor</td>
</tr>
<tr>
<td></td>
<td>penium3</td>
<td>Intel Pentium 3 style processor</td>
</tr>
<tr>
<td></td>
<td>penium4</td>
<td>Intel Pentium 4 style processor</td>
</tr>
<tr>
<td></td>
<td>amdfam10</td>
<td>AMD AMDFAM10 processor</td>
</tr>
</tbody>
</table>

### Defaults

On most processors, generic is the default value that directs the compiler to use the best timing properties for good performance without major performance degradation on any of the processors.

### A.2.119 —xcode=a

SPARC: Specifies the code address space.

**Note** — You should build shared objects by specifying -xcode=pic13 or -xcode=pic32. Shared objects built without pic13 or pic32 will not work correctly, and might not build at all.
A.2.119.1 Values

a must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs32</td>
<td>Generates 32-bit absolute addresses, which are fast, but have limited range. Code + data + bss size is limited to $2^{32}$ bytes.</td>
</tr>
<tr>
<td>abs44</td>
<td>SPARC: Generates 44-bit absolute addresses, which have moderate speed and moderate range. Code + data + bss size is limited to $2^{44}$ bytes. Available only on 64-bit architectures. Do not use this value with dynamic (shared) libraries.</td>
</tr>
<tr>
<td>abs64</td>
<td>SPARC: Generates 64-bit absolute addresses, which are slow, but have full range. Available only on 64-bit architectures.</td>
</tr>
<tr>
<td>pic13</td>
<td>Generates position-independent code (small model), which is fast, but has limited range. Equivalent to --Kpic. Permits references to at most $2^{11}$ unique external symbols on 32-bit architectures; $2^{10}$ on 64-bit.</td>
</tr>
<tr>
<td>pic32</td>
<td>Generates position-independent code (large model), which might not be as fast as pic13, but has full range. Equivalent to --KPIC. Permits references to at most $2^{30}$ unique external symbols on 32-bit architectures; $2^{29}$ on 64-bit.</td>
</tr>
</tbody>
</table>

To determine whether to use -xcode=pic13 or -xcode=pic32, check the size of the Global Offset Table (GOT) by using elfdump -c (see the elfdump(1) man page for more information) and for the section header, sh name: .got. The sh size value is the size of the GOT. If the GOT is less than 8,192 bytes, specify -xcode=pic13, otherwise specify -xcode=pic32.

In general, use the following guidelines to determine how you should use -xcode:

- If you are building an executable you should not use -xcode=pic13 or -xcode=pic32.
- If you are building an archive library only for linking into executables you should not use -xcode=pic13 or -xcode=pic32.
- If you are building a shared library, start with -xcode=pic13 and once the GOT size exceed 8,192 bytes, use -xcode=pic32.
- If you are building an archive library for linking into shared libraries you should just use -xcode=pic32.

Defaults

The default is -xcode=abs32 for 32-bit architectures. The default for 64-bit architectures is -xcode=abs44.
When building shared dynamic libraries, the default -xcode values of abs44 and abs32 will not work with 64-bit architectures. Specify -xcode=pic13 or -xcode=pic32 instead. There are two nominal performance costs with -xcode=pic13 and -xcode=pic32 on SPARC:

- A routine compiled with either -xcode=pic13 or -xcode=pic32 executes a few extra instructions upon entry to set a register to point at a table (_GLOBAL_OFFSET_TABLE_) used for accessing a shared library's global or static variables.
- Each access to a global or static variable involves an extra indirect memory reference through _GLOBAL_OFFSET_TABLE_. If the compile is done with -xcode=pic32, there are two additional instructions per global and static memory reference.

When considering the above costs, remember that the use of -xcode=pic13 and -xcode=pic32 can significantly reduce system memory requirements, due to the effect of library code sharing. Every page of code in a shared library compiled -xcode=pic13 or -xcode=pic32 can be shared by every process that uses the library. If a page of code in a shared library contains even a single non-pic (that is, absolute) memory reference, the page becomes nonsharable, and a copy of the page must be created each time a program using the library is executed.

The easiest way to tell whether or not a .o file has been compiled with -xcode=pic13 or -xcode=pic32 is with the nm command:

```
% nm file.o | grep _GLOBAL_OFFSET_TABLE_ U _GLOBAL_OFFSET_TABLE_
```

A .o file containing position-independent code contains an unresolved external reference to _GLOBAL_OFFSET_TABLE_, as indicated by the letter U.

To determine whether to use -xcode=pic13 or -xcode=pic32, use nm to identify the number of distinct global and static variables used or defined in the library. If the size of _GLOBAL_OFFSET_TABLE_ is under 8,192 bytes, you can use -Kpic. Otherwise, you must use -xcode=pic32.

**A.2.120**
-xcrossfile[=n]

Obsolete, do not use. Use -xipo instead.

**A.2.121**
-xdebugformat=[stabs|dwarf]

The compiler is migrating the format of debugger information from the stabs format to the dwarf format as specified in 'DWARF Debugging Information Format'. The default setting for this release is -xdebugformat=stabs.

If you maintain software which reads debugging information, you now have the option to transition your tools from the stabs format to the dwarf format.
Use this option as a way of accessing the new format for the purpose of porting tools. There is no need to use this option unless you maintain software which reads debugger information, or unless a specific tool tells you that it requires debugger information in one of these formats.

**TABLE A–30**  The -xdebugformat Flags

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>stabs</td>
<td>-xdebugformat=stabs generates debugging information using the stabs standard format.</td>
</tr>
<tr>
<td>dwarf</td>
<td>-xdebugformat=dwarf generates debugging information using the dwarf standard format.</td>
</tr>
</tbody>
</table>

If you do not specify -xdebugformat, the compiler assumes -xdebugformat=stabs. This option requires an argument.

This option affects the format of the data that is recorded with the -g option. Some small amount of debugging information is recorded even without -g, and the format of that information is also controlled with this option. So -xdebugformat has an effect even when -g is not used.

The dbx and Performance Analyzer software understand both stabs and dwarf format so using this option does not have any effect on the functionality of either tool.

---

**Note** – This is a transitional interface so expect it to change in incompatible ways from release to release, even in a minor release. The details of any specific fields or values in either stabs or dwarf are also evolving.

See also the dumpstabs(1) and dwarfdump(1) man pages for more information.

---

### A.2.122  -xdepend=[yes|no]

(SPARC) Analyzes loops for inter-iteration data dependencies and does loop restructuring, including loop interchange, loop fusion, scalar replacement, and elimination of “dead array” assignments.

On SPARC, -xdepend defaults to -xdepend=on for all optimization levels -xO3 and above. Otherwise -xdepend defaults to -xdepend=off. Specifying an explicit setting of -xdepend overrides any default setting.

On x86, --xdepend defaults to --xdepend=off. When --xdepend is specified and optimization is not at --xO3 or higher, the compiler raises the optimization to --xO3 and issues a warning.

Specifying -xdepend without an argument is equivalent to -xdepend=yes.
Dependency analysis is included in `-xautopar`. Dependency analysis is done at compile time.

Dependency analysis may help on single-processor systems. However, if you use `-xdepend` on single-processor systems, you should not also specify `-xautopar`, because the `-xdepend` optimization will be done for a multiprocessor system.

See Also: `-xprefetch_auto_type`

### A.2.123 `-xdumpmacros`\([=value[, value...]]\]

Use this option when you want to see how macros are behaving in your program. This option provides information such as macro defines, undefines, and instances of usage. It prints output to the standard error (`stderr`), based on the order macros are processed. The `-xdumpmacros` option is in effect through the end of the file or until it is overridden by the `dumpmacros` or `end_dumpmacros` pragma. See “B.2.5 #pragma dumpmacros” on page 366.

#### A.2.123.1 Values

You can substitute the following arguments in place of `value`:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[no]</code>defs</td>
<td>[Do not] Print all macro defines</td>
</tr>
<tr>
<td><code>[no]</code>undefs</td>
<td>[Do not] Print all macro undefines</td>
</tr>
<tr>
<td><code>[no]</code>use</td>
<td>[Do not] Print information about macros used</td>
</tr>
<tr>
<td><code>[no]</code>loc</td>
<td>[Do not] Print location (path name and line number) also for <code>defs</code>, <code>undefs</code>, and <code>use</code></td>
</tr>
<tr>
<td><code>[no]</code>conds</td>
<td>[Do not] Print use information for macros used in conditional directives</td>
</tr>
<tr>
<td><code>[no]</code>sys</td>
<td>[Do not] Print all macros defines, undefines, and use information for macros in system header files</td>
</tr>
<tr>
<td><code>%all</code></td>
<td>Sets the option to <code>-xdumpmacros=defs, undefs, use, loc, conds, sys</code>. A good way to use this argument is in conjunction with the <code>[no]</code> form of the other arguments. For example, <code>-xdumpmacros=%all, [no]sys</code> would exclude system header macros from the output but still provide information for all other macros.</td>
</tr>
<tr>
<td><code>%none</code></td>
<td>Do not print any macro information</td>
</tr>
</tbody>
</table>

The option values accumulate so specifying `-xdumpmacros=sys -xdumpmacros=undefs` has the same effect as `-xdumpmacros=undefs, sys`. 

---

296 Sun Studio 12 Update 1: C++ User's Guide
Note – The sub-options loc, conds, and sys are qualifiers for defs, undefs and use options. By themselves, loc, conds, and sys have no effect. For example, -xdumpmacros=loc,conds,sys has no effect.

Defaults

If you specify -xdumpmacros without any arguments, it means -xdumpmacros=defs,undefs,sys. If you do not specify -xdumpmacros, it defaults to -xdumpmacros=%none.

Examples

If you use the option -xdumpmacros=use,no%loc, the name of each macro that is used is printed only once. However, if you want more detail, use the option -xdumpmacros=use,loc so the location and macro name is printed every time a macro is used.

Consider the following file t.c:

```
example% cat t.c
#ifdef FOO
#undef FOO
#define COMPUTE(a, b) a+b
#else
#define COMPUTE(a,b) a-b
#endif
int n = COMPUTE(5,2);
int j = COMPUTE(7,1);
#if COMPUTE(8,3) + NN + MM
int k = 0;
#endif
```

The following examples show the output for file t.c based on the defs, undefs, sys, and loc arguments.

```
example% CC -c -xdumpmacros -DFOO t.c
#define __SunOS_5_9 1
#define __SUNPRO_CC 0x590
#define unix 1
#define sun 1
#define sparc 1
#define __sparc 1
#define __unix 1
#define __sun 1
#define __BUILTIN_VA_ARG_INCR 1
#define __SVR4 1
```
#define __SUNPRO_CC_COMPAT 5
#define __SUN_PREFETCH 1
#define FOO 1
#undef FOO
#define COMPUTE(a, b) a + b
example% CC -c -xdumpmacros=defs,undefs,loc -DFOO -UBAR t.c
command line: #define __SunOS_5_9 1
command line: #define __SUNPRO_CC 0x590
command line: #define unix 1
command line: #define sun 1
command line: #define sparc 1
command line: #define __sparc 1
command line: #define __UNIX 1
command line: #define __sun 1
command line: #define __BUILTIN_VA_ARG_INCR 1
command line: #define __SVR4 1
command line: #define __SUNPRO_CC_COMPAT 5
command line: #define __SUN_PREFETCH 1
command line: #define FOO 1
command line: #define BAR
T.c, line 2: #undef FOO
T.c, line 3: #define COMPUTE(a, b) a + b

The following examples show how the use, loc, and conds arguments report macro behavior in file t.c:

eexample% CC -c -xdumpmacros=use t.c
used macro COMPUTE

eexample% CC -c -xdumpmacros=use,loc t.c
T.c, line 7: used macro COMPUTE
T.c, line 8: used macro COMPUTE

eexample% CC -c -xdumpmacros=use,conds t.c
used macro FOO
used macro COMPUTE
used macro NN
used macro MM

eexample% CC -c -xdumpmacros=use,conds,loc t.c
T.c, line 1: used macro FOO
T.c, line 7: used macro COMPUTE
T.c, line 8: used macro COMPUTE
T.c, line 9: used macro COMPUTE
T.c, line 9: used macro NN
T.c, line 9: used macro MM

Consider the file y.c:
example% cat y.c
#define X 1
#define Y X
#define Z Y
int a = Z;

Here is the output from -xdumpmacros=use,loc based on the macros in y.c:

example% CC -c -xdumpmacros=use,loc y.c
y.c, line 4: used macro Z
y.c, line 4: used macro Y
y.c, line 4: used macro X

See Also
Use the dumpmacros pragma and the end_dumpmacros pragma when you want to override the scope of -xdumpmacros.

A.2.124 -xe
Checks only for syntax and semantic errors. When you specify -xe, the compiler does not produce any object code. The output for -xe is directed to stderr.

Use the -xe option if you do not need the object files produced by compilation. For example, if you are trying to isolate the cause of an error message by deleting sections of code, you can speed the edit and compile cycle by using -xe.

A.2.124.1 See Also
--c

A.2.125 -xF[=v[,...]]
Enables optimal reordering of functions and variables by the linker.

This option instructs the compiler to place functions and/or data variables into separate section fragments, which enables the linker, using directions in a mapfile specified by the linker's -M option, to reorder these sections to optimize program performance. Generally, this optimization is only effective when page fault time constitutes a significant fraction of program run time.

Reordering of variables can help solve the following problems which negatively impact run-time performance:

- Cache and page contention caused by unrelated variables that are near each other in memory.
Unnecessarily large work-set size as a result of related variables which are not near each other in memory.

Unnecessarily large work-set size as a result of unused copies of weak variables that decrease the effective data density.

Reordering variables and functions for optimal performance requires the following operations:

1. Compiling and linking with -xF.
2. Following the instructions in the “Program Performance Analysis Tools” manual regarding how to generate a mapfile for functions or following the instructions in the “Linker and Libraries Guide” regarding how to generate a mapfile for data.
3. Relinking with the new mapfile by using the linker’s -M option.
4. Re-executing under the Analyzer to verify improvement.

A.2.125.1 Values

_v can be one or more of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]func</td>
<td>[Do not] fragment functions into separate sections.</td>
</tr>
<tr>
<td>[no%]gbldata</td>
<td>[Do not] fragment global data (variables with external linkage) into separate sections.</td>
</tr>
<tr>
<td>[no%]lcldata</td>
<td>[Do not] fragment local data (variables with internal linkage) into separate sections.</td>
</tr>
<tr>
<td>%all</td>
<td>Fragment functions, global data, and local data.</td>
</tr>
<tr>
<td>%none</td>
<td>Fragment nothing.</td>
</tr>
</tbody>
</table>

Defaults

If you do not specify -xF, the default is -xF=%none. If you specify -xF without any arguments, the default is -xF=%none, func.

Interactions

Using -xF=lcldata inhibits some address calculation optimizations, so you should only use this flag when it is experimentally justified.

See also

analyzer(1), debugger(1), ld(1) man pages
A.2.126  \texttt{-xhelp=flags}
Displays a brief description of each compiler option.

A.2.127  \texttt{-xhelp=readme}
Displays contents of the online \texttt{readme} file.

The \texttt{readme} file is paged by the command specified in the environment variable, \texttt{PAGER}. If \texttt{PAGER}
is not set, the default paging command is \texttt{more}.

A.2.128  \texttt{-xhwcprof}
(SPARC) Enables compiler support for hardware counter-based profiling.

When \texttt{-xhwcprof} is enabled, the compiler generates information that helps tools associate
profiled load and store instructions with the data-types and structure members (in conjunction
with symbolic information produced with \texttt{-g} to which they refer. It associates profile data with
the data space of the target, rather than the instruction space, and provides insight into behavior
that is not easily obtained from only instruction profiling.

You can compile a specified set of object files with \texttt{-xhwcprof}. However, \texttt{-xhwcprof} is most
useful when applied to all object files in the application. This will provide coverage to identify
and correlate all memory references distributed in the application's object files.

If you are compiling and linking in separate steps, use \texttt{-xhwcprof} at link time as well. Future
extensions to \texttt{-xhwcprof} may require its use at link time.

An instance of \texttt{-xhwcprof=enable} or \texttt{-xhwcprof=disable} overrides all previous instances of
\texttt{-xhwcprof} in the same command line.

\texttt{-xhwcprof} is disabled by default. Specifying \texttt{-xhwcprof} without any arguments is the
equivalent to \texttt{-xhwcprof=enable}.

\texttt{-xhwcprof} requires that optimization be turned on and that the debug data format be set to
\texttt{DWARF(-xdebugformat=dwarf)}.

The combination of \texttt{-xhwcprof} and \texttt{-g} increases compiler temporary file storage requirements
by more than the sum of the increases due to \texttt{-xhwcprof} and \texttt{-g} specified alone.

The following command compiles \texttt{example.cc} and specifies support for hardware counter
profiling and symbolic analysis of data types and structure members using \texttt{DWARF} symbols:

\texttt{example\% CC -c -O -xhwcprof -g -xdebugformat=dwarf example.cc}
For more information on hardware counter-based profiling, see the Program Performance Analysis Tools manual.

A.2.129 -xia

Links the appropriate interval arithmetic libraries and sets a suitable floating-point environment.

**Note** – The C++ interval arithmetic library is compatible with interval arithmetic as implemented in the Fortran compiler.

On x86 platforms, requires support of SSE2 instruction set.

A.2.129.1 Expansions

The -xia option is a macro that expands to -fsimple=0 -ftrap=%none -fns=no -library=interval. If you use intervals and override what is set by -xia by specifying a different flag for -fsimple, -ftrap, -fns or -library, you may cause the compiler to exhibit incorrect behavior.

**Interactions**

To use the interval arithmetic libraries, include <suninterval.h>.

When you use the interval arithmetic libraries, you must include one of the following libraries: libc, Cstd, or iostreams. See -library for information on including these libraries.

**Warnings**

If you use intervals and you specify different values for -fsimple, -ftrap, or -fns, then your program may have incorrect behavior.

C++ interval arithmetic is experimental and evolving. The specifics may change from release to release.

**See Also**

-library

A.2.130 -xinline[=func_spec[,func_spec...]]

Specifies which user-written routines can be inlined by the optimizer at -x03 levels or higher.
A.2.130.1 Values

`func_spec` must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%auto</code></td>
<td>Enable automatic inlining at optimization levels <code>-xO4</code> or higher. This argument tells the optimizer that it can inline functions of its choosing. Note that without the <code>%auto</code> specification, automatic inlining is normally turned off when explicit inlining is specified on the command line by <code>-xinline=[no%]func_name...</code>.</td>
</tr>
<tr>
<td><code>func_name</code></td>
<td>Strongly request that the optimizer inline the function. If the function is not declared as extern &quot;C&quot;, the value of <code>func_name</code> must be mangled. You can use the <code>nm</code> command on the executable file to find the mangled function names. For functions declared as extern &quot;C&quot;, the names are not mangled by the compiler.</td>
</tr>
<tr>
<td><code>no%func_name</code></td>
<td>When you prefix the name of a routine on the list with <code>no%</code>, the inlining of that routine is inhibited. The rule about mangled names for <code>func_name</code> applies to <code>no%func_name</code> as well.</td>
</tr>
</tbody>
</table>

Only routines in the file being compiled are considered for inlining unless you use `-xipo=[1|2]`. The optimizer decides which of these routines are appropriate for inlining.

Defaults

If the `-xinline` option is not specified, the compiler assumes `-xinline=%auto`.

If `-xinline=` is specified with no arguments, no functions are inlined, regardless of the optimization level.

Examples

To enable automatic inlining while disabling inlining of the function declared `int foo()`, use

```
example% CC -xO5 -xinline=%auto,no%__1cDfoo6F_i_ -c a.cc
```

To strongly request the inlining of the function declared as `int foo()`, and to make all other functions as the candidates for inlining, use

```
example% CC -xO5 -xinline=%auto,__1cDfoo6F_i_ -c a.cc
```

To strongly request the inlining of the function declared as `int foo()`, and to not allow inlining of any other functions, use

```
example% CC -xO5 -xinline=__1cDfoo6F_i_ -c a.cc
```
Interactions

The -xinline option has no effect for optimization levels below -O3. At -O4 and higher, the optimizer decides which functions should be inlined, and does so without the -xinline option being specified. At -O4 and higher, the compiler also attempts to determine which functions will improve performance if they are inlined.

A routine is inlined if any of the following conditions apply.

- Optimization is -O3 or greater
- Inlining is judged to be profitable and safe
- The function is in the file being compiled, or the function is in a file that was compiled with -xipo [=1|2]

Warnings

If you force the inlining of a function with -xinline, you might actually diminish performance.

See Also

“A.2.136 -xldsconf=\{v\}” on page 310

A.2.131 -xinstrument=[no%]datarace

Specify this option to compile and instrument your program for analysis by the Thread Analyzer. For more information on the Thread Analyzer, see th(a) for details.

You can then use the Performance Analyzer to run the instrumented program with collect -r races to create a data-race-detection experiment. You can run the instrumented code standalone but it runs more slowly.

You can specify -xinstrument=no%datarace to turn off preparation of source code for the thread analyzer. This is the default.

It is illegal to specify -xinstrument without an argument.

If you compile and link in separate steps, you must specify -xinstrument=datarace in both the compilation and linking steps.

This option defines the preprocessor token __THA_NOTIFY. You can specify #ifdef __THA_NOTIFY to guard calls to libtha(3) routines.

This option also sets -g.
A.2.132 -xipo\{=0\(1\|2\}\}

Performs interprocedural optimizations.

The -xipo option performs partial-program optimizations by invoking an interprocedural analysis pass. It performs optimizations across all object files in the link step, and the optimizations are not limited to just the source files on the compile command. However, whole-program optimizations performed with -xipo do not include assembly (.s) source files.

The -xipo option is particularly useful when compiling and linking large multifile applications. Object files compiled with this flag have analysis information compiled within them that enables interprocedural analysis across source and precompiled program files. However, analysis and optimization is limited to the object files compiled with -xipo, and does not extend to object files or libraries.

A.2.132.1 Values

The -xipo option can have the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not perform interprocedural optimizations</td>
</tr>
<tr>
<td>1</td>
<td>Perform interprocedural optimizations</td>
</tr>
<tr>
<td>2</td>
<td>Perform interprocedural aliasing analysis as well as optimizations of memory allocation and layout to improve cache performance</td>
</tr>
</tbody>
</table>

Defaults

If -xipo is not specified, -xipo=0 is assumed.

If only -xipo is specified, -xipo=1 is assumed.

Examples

The following example compiles and links in the same step.

```
exampel% CC -xipo -xO4 -o prog part1.cc part2.cc part3.cc
```

The optimizer performs crossfile inlining across all three source files. This is done in the final link step, so the compilation of the source files need not all take place in a single compilation and could be over a number of separate compilations, each specifying the -xipo option.

The following example compiles and links in separate steps.
The object files created in the compile steps have additional analysis information compiled within them to permit crossfile optimizations to take place at the link step.

**Interactions**

The -xipo option requires at least optimization level -xO4.

You cannot use both the -xipo option and the -xcrossfile option in the same compiler command line.

**Warnings**

When compiling and linking are performed in separate steps, -xipo must be specified in both steps to be effective.

Objects that are compiled without -xipo can be linked freely with objects that are compiled with -xipo.

Libraries do not participate in crossfile interprocedural analysis, even when they are compiled with -xipo, as shown in this example.

```
example% CC -xipo -xO4 one.cc two.cc three.cc
example% CC -xar -o mylib.a one.o two.o three.o
...
example% CC -xipo -xO4 -o myprog main.cc four.cc mylib.a
```

In this example, interprocedural optimizations will be performed between one.cc, two.cc and three.cc, and between main.cc and four.cc, but not between main.cc or four.cc and the routines in mylib.a. (The first compilation may generate warnings about undefined symbols, but the interprocedural optimizations will be performed because it is a compile and link step.)

The -xipo option generates significantly larger object files due to the additional information needed to perform optimizations across files. However, this additional information does not become part of the final executable binary file. Any increase in the size of the executable program will be due to the additional optimizations performed.

**A.2.132.2 When Not To Use -xipo Interprocedural Analysis**

The compiler tries to perform whole-program analysis and optimizations as it works with the set of object files in the link step. The compiler makes the following two assumptions for any function (or subroutine) foo() defined in this set of object files:

- foo() is not called explicitly by another routine that is defined outside this set of object files at runtime.
The calls to foo() from any routine in the set of object files are not interposed upon by a different version of foo() defined outside this set of object files.

Do not compile with -xipo=2, if assumption 1 is not true for the given application.

Do not compile with either -xipo=1 or -xipo=2, if assumption 2 is not true.

As an example, consider interposing on the function malloc() with your own version and compiling with -xipo=2. Consequently, all the functions in any library that reference malloc() that are linked with your code have to be compiled with -xipo=2 also and their object files need to participate in the link step. Since this might not be possible for system libraries, do not compile your version of malloc with -xipo=2.

As another example, suppose that you build a shared library with two external calls, foo() and bar() inside two different source files. Furthermore, suppose that bar() calls foo(). If there is a possibility that foo() could be interposed at runtime, then do not compile the source file for foo() or for bar() with -xipo=1 or -xipo=2. Otherwise, foo() could be inlined into bar(), which could cause incorrect results.

See Also
-xjobs

A.2.133 -xipo_archive=[a]

The -xipo_archive option enables the compiler to optimize object files that are passed to the linker with object files that were compiled with -xipo and that reside in the archive library (.a) before producing an executable. Any object files contained in the library that were optimized during the compilation are replaced with their optimized version.

a is one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>writeback</td>
<td>The compiler optimizes object files passed to the linker with object files compiled with -xipo that reside in the archive library (.a) before producing an executable. Any object files contained in the library that were optimized during the compilation are replaced with an optimized version. For parallel links that use a common set of archive libraries, each link should create its own copy of archive libraries to be optimized before linking.</td>
</tr>
</tbody>
</table>
**TABLE A–35  The-xipo_archive Flags (Continued)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>readonly</td>
<td>The compiler optimizes object files passed to the linker with object files compiled with -xipo that reside in the archive library (.a) before producing an executable.</td>
</tr>
<tr>
<td></td>
<td>The option -xipo_archive=readonly enables cross-module inlining and interprocedural data flow analysis of object files in an archive library specified at link time. However, it does not enable cross-module optimization of the archive library's code except for code that has been inserted into other modules by cross module inlining.</td>
</tr>
<tr>
<td></td>
<td>To apply cross-module optimization to code within an archive library, -xipo_archive=writeback is required. Note that doing so modifies the contents of the archive library from which the code was extracted.</td>
</tr>
<tr>
<td>none</td>
<td>This is the default. There is no processing of archive files. The compiler does not apply cross-module inlining or other cross-module optimizations to object files compiled using -xipo and extracted from an archive library at link time. To do that, both -xipo and either -xipo_archive=readonly or -xipo_archive=writeback must be specified at link time.</td>
</tr>
</tbody>
</table>

If you do not specify a setting for -xipo_archive, the compiler sets it to -xipo_archive=none.

It is illegal to specify -xipo_archive without a flag.

### A.2.134 -xjobs=n

Specify the -xjobs option to set how many processes the compiler creates to complete its work. This option can reduce the build time on a multi-cpu machine. Currently, -xjobs works only with the -xipo option. When you specify -xjobs=n, the interprocedural optimizer uses n as the maximum number of code generator instances it can invoke to compile different files.

#### A.2.134.1 Values

You must always specify -xjobs with a value. Otherwise an error diagnostic is issued and compilation aborts.

Generally, a safe value for n is 1.5 multiplied by the number of available processors. Using a value that is many times the number of available processors can degrade performance because of context switching overheads among spawned jobs. Also, using a very high number can exhaust the limits of system resources such as swap space.

#### Defaults

Multiple instances of -xjobs on the command line override each other until the right-most instance is reached.
Examples

The following example compiles more quickly on a system with two processors than the same command without the -xjobs option.

```
example% CC -xipo -xO4 -xjobs=3 t1.cc t2.cc t3.cc
```

A.2.135 -xlang=\texttt{language[, language]}

Includes the appropriate runtime libraries and ensures the proper runtime environment for the specified language.

A.2.135.1 Values

\texttt{language} must be either \texttt{f77}, \texttt{f90}, \texttt{f95}, or \texttt{c99}.

The \texttt{f90} and \texttt{f95} arguments are equivalent. The \texttt{c99} argument invokes ISO 9899:1999 C programming language behavior for objects that were compiled with \texttt{cc -xc99=%all} and are being linked with \texttt{CC}.

Interactions

The -xlang=f90 and -xlang=f95 options imply -library=f90, and the -xlang=f77 option implies -library=f77. However, the -library=f77 and -library=f90 options are not sufficient for mixed-language linking because only the -xlang option ensures the proper runtime environment.

To determine which driver to use for mixed-language linking, use the following language hierarchy:

1. C++
2. Fortran 95 (or Fortran 90)
3. Fortran 77
4. C or C99

When linking Fortran 95, Fortran 77, and C++ object files together, use the driver of the highest language. For example, use the following C++ compiler command to link C++ and Fortran 95 object files.

```
example% CC -xlang=f95 ...
```

To link Fortran 95 and Fortran 77 object files, use the Fortran 95 driver, as follows.

```
example% f95 -xlang=f77 ...
```

You cannot use the -xlang option and the -xlic_lib option in the same compiler command. If you are using -xlang and you need to link in the Sun Performance Libraries, use -library=sunperf instead.
Warnings
Do not use -xnolib with -xlang.

If you are mixing parallel Fortran objects with C++ objects, the link line must specify the -mt flag.

See also
-library, -staticlib

A.2.136 -xldscope={v}

Specify the -xldscope option to change the default linker scoping for the definition of extern symbols. Changing the default can result in faster and safer shared libraries and executables because the implementation are better hidden.

A.2.136.1 Values

v must be one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>global</td>
<td>Global linker scoping is the least restrictive linker scoping. All references to the symbol bind to the definition in the first dynamic load module that defines the symbol. This linker scoping is the current linker scoping for extern symbols.</td>
</tr>
<tr>
<td>symbolic</td>
<td>Symbolic linker scoping and is more restrictive than global linker scoping. All references to the symbol from within the dynamic load module being linked bind to the symbol defined within the module. Outside of the module, the symbol appears as though it is global. This linker scoping corresponds to the linker option -Bsymbolic. Although you cannot use -Bsymbolic with C++ libraries, you can use the -xldscope=symbolic without causing problems. See ld(1) for more information on the linker.</td>
</tr>
<tr>
<td>hidden</td>
<td>Hidden linker scoping is more restrictive than symbolic and global linker scoping. All references within a dynamic load module bind to a definition within that module. The symbol will not be visible outside of the module.</td>
</tr>
</tbody>
</table>

Defaults
If you do not specify -xldscope, the compiler assumes -xldscope=global. If you specify -xldscope without any values, the compiler issues an error. Multiple instances of this option on the command line override each other until the right most instance is reached.
Warning

If you intend to allow a client to override a function in a library, you must be sure that the function is not generated inline during the library build. The compiler inlines a function if you specify the function name with -xinline, if you compile at -xO4 or higher in which case inlining can happen automatically, if you use the inline specifier, or if you are using cross-file optimization.

For example, suppose library ABC has a default allocator function that can be used by library clients, and is also used internally in the library:

```c
void* ABC_allocator(size_t size) { return malloc(size); }
```

If you build the library at -xO4 or higher, the compiler inlines calls to ABC_allocator that occur in library components. If a library client wants to replace ABC_allocator with a customized version, the replacement will not occur in library components that called ABC_allocator. The final program will include different versions of the function.

Library functions declared with the __hidden or __symbolic specifiers can be generated inline when building the library. They are not supposed to be overridden by clients. See "4.1 Linker Scoping" on page 63.

Library functions declared with the __global specifier, should not be declared inline, and should be protected from inlining by use of the -xinline compiler option.

See Also

-xinline, -xO, -xcrossfile

A.2.137  
-xlibmieee

Causes libm to return IEEE 754 values for math routines in exceptional cases.

The default behavior of libm is XPG-compliant.

A.2.137.1  
See also

Numerical Computation Guide

A.2.138  
-xlibmil

Inlines selected libm library routines for optimization.
Note – This option does not affect C++ inline functions.

There are inline templates for some of the libm library routines. This option selects those inline templates that produce the fastest executables for the floating-point option and platform currently being used.

A.2.138.1 Interactions
This option is implied by the –fast option.

See also
- fast, Numerical Computation Guide

A.2.139 –xlibmopt
Uses library of optimized math routines. You must use default rounding mode by specifying -fround=nearest when you use this option.

This option uses a math routine library optimized for performance and usually generates faster code. The results might be slightly different from those produced by the normal math library; if so, they usually differ in the last bit.

The order on the command line for this library option is not significant.

A.2.139.1 Interactions
This option is implied by the –fast option.

See also
- fast, -xnolibmopt, -fround

A.2.140 –xlic_lib=sunperf
Deprecated, do not use. Specify -library=sunperf instead. See “A.2.50 -library=‖,‖…” on page 252 for more information.

A.2.141 –xlicinfo
This option is silently ignored by the compiler.
**A.2.142 -xlinkopt[=-level]**

Instructs the compiler to perform link-time optimization on the resulting executable or dynamic library over and above any optimizations in the object files. These optimizations are performed at link time by analyzing the object binary code. The object files are not rewritten but the resulting executable code may differ from the original object codes.

You must use `-xlinkopt` on at least some of the compilation commands for `-xlinkopt` to be useful at link time. The optimizer can still perform some limited optimizations on object binaries that are not compiled with `-xlinkopt`.

`-xlinkopt` optimizes code coming from static libraries that appear on the compiler command line, but it skips and does not optimize code coming from shared (dynamic) libraries that appear on the command line. You can also use `-xlinkopt` when you build shared libraries (compiling with `-G`).

### Values

`level` sets the level of optimizations performed, and must be 0, 1, or 2. The optimization levels are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The link optimizer is disabled. (This is the default.)</td>
</tr>
<tr>
<td>1</td>
<td>Perform optimizations based on control flow analysis, including instruction cache coloring and branch optimizations, at link time.</td>
</tr>
<tr>
<td>2</td>
<td>Perform additional data flow analysis, including dead-code elimination and address computation simplification, at link time.</td>
</tr>
</tbody>
</table>

If you compile in separate steps, `-xlinkopt` must appear on both compile and link steps:

```bash
eexample% cc -c -xlinkopt a.c b.c
eexample% cc -o myprog -xlinkopt=2 a.o
```

Note that the level parameter is only used when the compiler is linking. In the example above, the link optimizer level is 2 even though the object binaries are compiled with an implied level of 1.

### Defaults

Specifying `-xlinkopt` without a level parameter implies `-xlinkopt=1`. 
Interactions

This option is most effective when you use it to compile the whole program, and with profile feedback. Profiling reveals the most and least used parts of the code and building directs the optimizer to focus its effort accordingly. This is particularly important with large applications where optimal placement of code performed at link time can reduce instruction cache misses. Typically, this compiles as follows:

```
example% cc -o progt -xO5 -xprofile=collect:prog file.c
example% progt
example% cc -o prog -xO5 -xprofile=use:prog -xlinkopt file.c
```

For details on using profile feedback, see "A.2.170 -xprofile=p" on page 339.

Warnings

Do not use the -zcombreloc linker option when you compile with -xlinkopt.

Note that compiling with this option increases link time slightly. Object file sizes also increase, but the size of the executable remains the same. Compiling with -xlinkopt and -g increases the size of the executable by including debugging information.

A.2.143 -xloopinfo

This option shows which loops are parallelized and which are not. It is normally for use with the -xautopar option.

A.2.144 -xM

Runs only the C++ preprocessor on the named C++ programs, requesting that the preprocessor generate makefile dependencies and send the result to the standard output (see make(1) for details about make files and dependencies).

However, -xM only reports dependencies of the included headers and not the associated template definition files. You can use the .KEEP_STATE feature in your makefile to generate all the dependencies in the .make.state file which the make utility creates.

A.2.144.1 Examples

For example:

```c
#include <unistd.h>
void main(void)
{
}
```
generates this output:

e.o: e.c
e.o: /usr/include/unistd.h
e.o: /usr/include/sys/types.h
e.o: /usr/include/sys/machtypes.h
e.o: /usr/include/sys/select.h
e.o: /usr/include/sys/time.h
e.o: /usr/include/sys/types.h
e.o: /usr/include/sys/unistd.h

Interactions

If you specify -xM and -xMF, the compiler writes all makefile dependency information to the file specified with -xMF. This file is overwitten each time the preprocessor writes to it.

See also

make(1S) (for details about makefiles and dependencies)

A.2.145 -xM1

Generates makefile dependencies like -xM, except that it does not report dependencies for the /usr/include header files and it does not report dependencies for compiler-supplied header files.

If you specify -xM1 and -xMF, the compiler writes all makefile dependency information to the file specified with -xMF. This file is overwitten each time the preprocessor writes to it.

A.2.146 -xMD

Generates makefile dependencies like -xM but includes compilation. -xMD generates an output file for the makefile-dependency information based on the input filename but with the addition of a .d suffix. If you specify -xMD and -xMF, the preprocessor appends all makefile dependency information to the file specified with -xMF.

A.2.147 -xMF

Use this option to specify a file for the makefile-dependency output. There is no way to specify individual filenames for multiple input files with -xMF on the command line.
A.2.148  -xMMD

Use this option to generate makefile dependencies excluding system header files. This is the same functionality as -xM1, but includes compilation. -xMMD generates an output file for the makefile-dependency information based on the input filename but with the addition of a .d suffix. If you specify both -xMMD and -xMF, the compiler uses the filename you provide instead.

A.2.149  -xMerge

SPARC: Merges the data segment with the text segment.

The data in the object file is read-only and is shared between processes, unless you link with ld -N.

The three options -xMerge -ztext -xprofile=collect should not be used together. While -xMerge forces statically initialized data into read-only storage, -ztext prohibits position-dependent symbol relocations in read-only storage, and -xprofile=collect generates statically initialized, position-dependent symbol relocations in writable storage.

A.2.149.1  See also

ld(1) man page

A.2.150  -xmaxopt[=v]

This command limits the level of pragma opt to the level specified. v is one of off, 1, 2, 3, 4, 5. The default value is -xmaxopt=off which causes pragma opt to be ignored. If you specify -xmaxopt without supplying an argument, that is the equivalent of specifying -xmaxopt=5.

If you specify both -xO and -xmaxopt, the optimization level set with -xO must not exceed the -xmaxopt value.

A.2.151  -xmemalign=ab

(SPARC) Use the -xmemalign option to control the assumptions the compiler makes about the alignment of data. By controlling the code generated for potentially misaligned memory accesses and by controlling program behavior in the event of a misaligned access, you can more easily port your code to SPARC.

Specify the maximum assumed memory alignment and behavior of misaligned data accesses. There must be a value for both a (alignment) and b (behavior). a specifies the maximum assumed memory alignment and b specifies the behavior for misaligned memory accesses.
For memory accesses where the alignment is determinable at compile time, the compiler generates the appropriate load/store instruction sequence for that alignment of data.

For memory accesses where the alignment cannot be determined at compile time, the compiler must assume an alignment to generate the needed load/store sequence.

If actual data alignment at runtime is less than the specified alignment, the misaligned access attempt (a memory read or write) generates a trap. The two possible responses to the trap are

- The OS converts the trap to a SIGBUS signal. If the program does not catch the signal, the program aborts. Even if the program catches the signal, the misaligned access attempt will not have succeeded.
- The OS handles the trap by interpreting the misaligned access and returning control to the program as if the access had succeeded normally.

### A.2.151.1 Values

The following table lists the alignment and behavior values for `-xmemalign`.

<table>
<thead>
<tr>
<th><code>a</code></th>
<th><code>b</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>i</code></td>
<td>Assume at most 1 byte alignment. Interpret access and continue execution.</td>
</tr>
<tr>
<td>2</td>
<td><code>s</code></td>
<td>Assume at most 2 byte alignment. Raise signal SIGBUS.</td>
</tr>
<tr>
<td>4</td>
<td><code>f</code></td>
<td>Assume at most 4 byte alignment. For variants of <code>-xarch=v9</code> only: Raise signal SIGBUS for alignments less or equal to 4, otherwise interpret access and continue execution. For all other <code>-xarch</code> values, the <code>f</code> flag is equivalent to <code>i</code>.</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Assume at most 8 byte alignment.</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>Assume at most 16 byte alignment.</td>
</tr>
</tbody>
</table>

You must specify `-xmemalign` whenever you want to link to an object file that was compiled with the value of `b` set to either `i` or `f`. For a complete list of all compiler options that must be specified at both compile time and at link time, see “3.3.3 Compile-Time and Link-Time Options” on page 50.

### Defaults

The following default values only apply when no `-xmemalign` option is present:

- `-xmemalign=8i` for all v8 architectures.
- `-xmemalign=8s` for all v9 architectures.
Here is the default when the `-xmemalign` option is present but no value is given:

- `-xmemalign=1i` for all `-xarch` values.

**Examples**

The following table shows how you can use `-xmemalign` to handle different alignment situations.

<table>
<thead>
<tr>
<th>Command</th>
<th>Situation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-xmemalign=1s</code></td>
<td>All memory accesses are misaligned so trap handling is too slow.</td>
</tr>
<tr>
<td><code>-xmemalign=8i</code></td>
<td>There are occasional, intentional, misaligned accesses in code that is otherwise correct.</td>
</tr>
<tr>
<td><code>-xmemalign=8s</code></td>
<td>There should be no misaligned accesses in the program.</td>
</tr>
<tr>
<td><code>-xmemalign=2s</code></td>
<td>You want to check for possible odd-byte accesses.</td>
</tr>
<tr>
<td><code>-xmemalign=2i</code></td>
<td>You want to check for possible odd-byte access and you want the program to work.</td>
</tr>
</tbody>
</table>

**A.2.152**

`-xmodel=[a]`  
(x86) The `-xmodel` option enables the compiler to modify the form of 64-bit objects for the Solaris x86 platforms and should only be specified for the compilation of such objects.

This option is valid only when `-m64` is also specified on 64–bit enabled x64 processors.

`a` must be one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>This option generates code for the small model in which the virtual address of code executed is known at link time and all symbols are known to be located in the virtual addresses in the range from 0 to $2^{31} - 2^{24} - 1$.</td>
</tr>
<tr>
<td>kernel</td>
<td>Generates code for the kernel model in which all symbols are defined to be in the range from $2^{64} - 2^{31}$ to $2^{64} - 2^{24}$.</td>
</tr>
</tbody>
</table>
The \texttt{-xmodel} Flags (Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>medium</td>
<td>Generates code for the medium model in which no assumptions are made about the range of symbolic references to data sections. Size and address of the text section have the same limits as the small code model. Applications with large amounts of static data might require \texttt{-xmodel=medium} when compiling with \texttt{-m64}.</td>
</tr>
</tbody>
</table>

This option is not cumulative so the compiler sets the model value according to the rightmost instance of \texttt{-xmodel} on the command-line.

If you do not specify \texttt{-xmodel}, the compiler assumes \texttt{-xmodel=small}. Specifying \texttt{-xmodel} without an argument is an error.

It is not necessary to compile all translation units with this option. You can compile select files as long as you ensure the object you are accessing is within reach.

Be aware that not all Linux systems support the medium model.

\textbf{A.2.153} \texttt{-xnolib}

Disables linking with default system libraries.

Normally (without this option), the C++ compiler links with several system support libraries to support C++ programs. With this option, the \texttt{-l} options to link the default system support libraries are not passed to \texttt{ld}.

Normally, the compiler links with the system support libraries in the following order:

- Standard mode (default mode):
  - \texttt{-lCstd} \texttt{-lCrun} \texttt{-lm} \texttt{-lc}

- Compatibility mode (\texttt{-compat}):
  - \texttt{-lC} \texttt{-lm} \texttt{-lc}

The order of the \texttt{-l} options is significant. The \texttt{-lm} option must appear before \texttt{-lc}.

\textbf{Note} - If the \texttt{-mt} compiler option is specified, the compiler normally links with \texttt{-lthread} just before it links with \texttt{-lm}.

To determine which system support libraries will be linked by default, compile with the \texttt{-dryrun} option. For example, the output from the following command:
A.2 Option Reference

example% CC foo.cc -xarch=v9 -dryrun

Includes the following in the output:

-ldstd -lCrun -lm -lc

A.2.153.1 Examples

For minimal compilation to meet the C application binary interface (that is, a C++ program
with only C support required), use:

example% CC -xnolib test.cc -lc

To link libm statically into a single-threaded application with the generic architecture
instruction set, use:

Interactions

If you specify -xnolib, you must manually link all required system support libraries in the
given order. You must link the system support libraries last.

If -xnolib is specified, -library is ignored.

Warnings

Many C++ language features require the use of libC (compatibility mode) or libCrun
(standard mode).

This set of system support libraries is not stable and might change from release to release.

See also

-library, -staticlib, -l

A.2.154 -xnolibmil

Cancels -xlibmil on the command line.

Use this option with -fast to override linking with the optimized math library.

A.2.155 -xnolibmopt

Does not use the math routine library.
A.2.155.1 Examples
Use this option after the –fast option on the command line, as in this example:

```
example% CC –fast –xnolibmopt
```

A.2.156 -xnorunpath
Same as "A.2.63 –norunpath" on page 259

A.2.157 -xO level
Specifies optimization level; note the uppercase letter O followed by the digit 1, 2, 3, 4, or 5. In general, program execution speed depends on the level of optimization. The higher the level of optimization, the better the runtime performance. However, higher optimization levels can result in increased compilation time and larger executable files.

In a few cases, –xO2 might perform better than the others, and –xO3 might outperform –xO4. Try compiling with each level to see if you have one of these rare cases.

If the optimizer runs out of memory, it tries to recover by retrying the current procedure at a lower level of optimization. The optimizer resumes subsequent procedures at the original level specified in the -xO level option.

There are five levels that you can use with –xO. The following sections describe how they operate on the SPARC platform and the x86 platform.

A.2.157.1 Values
On the SPARC Platform:
- –xO1 does only the minimum amount of optimization (peephole), which is post-pass, assembly-level optimization. Do not use -xO1 unless using -xO2 or -xO3 results in excessive compilation time, or you are running out of swap space.
- –xO2 does basic local and global optimization, which includes:
  - Induction-variable elimination
  - Local and global common-subexpression elimination
  - Algebraic simplification
  - Copy propagation
  - Constant propagation
  - Loop-invariant optimization
  - Register allocation
### A.2 Option Reference

- Basic block merging
- Tail recursion elimination
- Dead-code elimination
- Tail-call elimination
- Complicated expression expansion

This level does not optimize references or definitions for external or indirect variables.

`-xO3`, in addition to optimizations performed at the `--xO2` level, also optimizes references and definitions for external variables. This level does not trace the effects of pointer assignments. When compiling either device drivers that are not properly protected by `volatile` or programs that modify external variables from within signal handlers, use `--xO2`. In general, this level results in increased code size unless combined with the `-xspace` option.

- `--xO4` does automatic inlining of functions contained in the same file in addition to performing `--xO3` optimizations. This automatic inlining usually improves execution speed but sometimes makes it worse. In general, this level results in increased code size unless combined with the `-xspace` option.

- `--xO5` generates the highest level of optimization. It is suitable only for the small fraction of a program that uses the largest fraction of computer time. This level uses optimization algorithms that take more compilation time or that do not have as high a certainty of improving execution time. Optimization at this level is more likely to improve performance if it is done with profile feedback. See "A.2.170 `-xprofile`" on page 339.

### On the x86 Platform:

- `--xO1` does basic optimization. This includes algebraic simplification, register allocation, basic block merging, dead code and store elimination, and peephole optimization.
- `--xO2` performs local common subexpression elimination, local copy and constant propagation, and tail recursion elimination, as well as the optimization done by level 1.
- `--xO3` performs global common subexpression elimination, global copy and constant propagation, loop strength reduction, induction variable elimination, and loop-variant optimization, as well as the optimization done by level 2.
- `--xO4` does automatic inlining of functions contained in the same file as well as the optimization done by level 3. This automatic inlining usually improves execution speed, but sometimes makes it worse. This level also frees the frame pointer registration (ebp) for general purpose use. In general this level results in increased code size.
- `--xO5` generates the highest level of optimization. It uses optimization algorithms that take more compilation time or that do not have as high a certainty of improving execution time.
Interactions

If you use -g or -g0 and the optimization level is -xO3 or lower, the compiler provides best-effort symbolic information with almost full optimization. Tail-call optimization and back-end inlining are disabled.

If you use -g or -g0 and the optimization level is -xO4 or higher, the compiler provides best-effort symbolic information with full optimization.

Debugging with -g does not suppress -xO level, but -xO level limits -g in certain ways. For example, the -xO level options reduce the utility of debugging so that you cannot display variables from dbx, but you can still use the dbx where command to get a symbolic traceback. For more information, see Debugging a Program With dbx.

The -xipo option is effective only if it is used with -xO4 or -xO5.

The -xinline option has no effect for optimization levels below -xO3. At -xO4, the optimizer decides which functions should be inlined, and does so regardless of whether you specify the -xinline option. At -xO4, the compiler also attempts to determine which functions will improve performance if they are inlined. If you force the inlining of a function with -xinline, you might actually diminish performance.

Defaults

The default is no optimization. However, this is only possible if you do not specify an optimization level. If you specify an optimization level, there is no option for turning optimization off.

If you are trying to avoid setting an optimization level, be sure not to specify any option that implies an optimization level. For example, -fast is a macro option that sets optimization at -xO5. All other options that imply an optimization level give a warning message that optimization has been set. The only way to compile without any optimization is to delete all options from the command line or make file that specify an optimization level.

Warnings

If you optimize at -xO3 or -xO4 with very large procedures (thousands of lines of code in a single procedure), the optimizer might require an unreasonable amount of memory. In such cases, machine performance can be degraded.

To prevent this degradation from taking place, use the limit command to limit the amount of virtual memory available to a single process (see the csh(1) man page). For example, to limit virtual memory to 4 gigabytes:

c example% limit datasize 4G

This command causes the optimizer to try to recover if it reaches 4 gigabytes of data space.
The limit cannot be greater than the total available swap space of the machine, and should be small enough to permit normal use of the machine while a large compilation is in progress.

The best setting for data size depends on the degree of optimization requested, the amount of real memory, and virtual memory available.

To find the actual swap space, type: `swap –l`

To find the actual real memory, type: `dmesg | grep mem`

**See also**
-xldscope –fast, -xcrossfile=n, –xprofile=p, csh(1) man page

### A.2.158 `-xopenmp[=i]`

Use the `-xopenmp` option to enable explicit parallelization with OpenMP directives. To run a parallelized program in a multithreaded environment, you must set the `OMP_NUM_THREADS` environment variable prior to execution.

To enable nested parallelism, you must set the `OMP_NESTED` environment variable to TRUE. Nested parallelism is disabled by default.

### A.2.158.1 Values

The following table lists the values for `i`:

<table>
<thead>
<tr>
<th>Values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel</td>
<td>Enables recognition of OpenMP pragmas. The minimum optimization level under <code>-xopenmp=parallel</code> is <code>-O3</code>. The compiler changes the optimization from a lower level to <code>-O3</code> if necessary and issues a warning. This flag also defines the preprocessor token <code>_OPENMP</code>.</td>
</tr>
<tr>
<td>noopt</td>
<td>Enables recognition of OpenMP pragmas. The compiler does not raise the optimization level if it is lower than <code>-O3</code>. If you explicitly set the optimization lower than <code>-O3</code>, as in <code>CC -O2 -xopenmp=noopt</code>, the compiler issues an error. If you do not specify an optimization level with <code>-xopenmp=noopt</code>, the OpenMP pragmas are recognized, the program is parallelized accordingly, but no optimization is done. This flag also defines the preprocessor token <code>_OPENMP</code>.</td>
</tr>
</tbody>
</table>
TABLE A-41 The -xopenmp Values (Continued)

<table>
<thead>
<tr>
<th>Values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>This flag is the default and disables recognition of OpenMP pragmas, does not change the optimization level of your program, and does not predefine any preprocessor tokens.</td>
</tr>
</tbody>
</table>

**Defaults**

If you do not specify `-xopenmp`, the compiler sets the option to `-xopenmp=none`.

If you specify `-xopenmp`, but without an argument, the compiler sets the option to `-xopenmp=parallel`.

**Interactions**

If you are debugging an OpenMP program with dbx, compile with `-g` and `-xopenmp=noopt` so you can breakpoint within parallel regions and display the contents of variables.

**Warnings**

Do not specify `-xopenmp`, with either `-xexplicitpar`, or `-xparallel`.

The default for `-xopenmp` might change in future releases. You can avoid warning messages by explicitly specifying an appropriate optimization.

If you compile and link in separate steps, specify `-xopenmp` in both the compilation step and the link step. This is important if you are building a shared object. The compiler which was used to compile the executable must not be any older than the compiler that built the .so with `-xopenmp`. This is especially important when you compile libraries that contain OpenMP directives. See "3.3.3 Compile-Time and Link-Time Options" on page 50 for a complete list of options that must be specified at both compile time and link time.

Make sure that the latest patch of the OpenMP runtime library, `libmtsk.so`, is installed on the system for best performance.

**See also**

For a complete summary of the OpenMP Fortran 95, C, and C++ application program interface (API) for building multiprocessing applications, see the *Sun Studio OpenMP API User’s Guide*.

**A.2.159 -xpagesize=n**

Sets the preferred page size for the stack and the heap.
A.2.159.1  **Values**  
The following values are valid for SPARC: 4k, 8K, 64K, 512K, 2M, 4M, 32M, 256M, 2G, 16G, or default.

The following values are valid on x86/x64: 4K, 2M, 4M, 1G, or default.

You must specify a valid page size for the target platform. If you do not specify a valid pagesize, the request is silently ignored at run-time.

Use the `getpagesize(3C)` command on the Solaris operating system to determine the number of bytes in a page. The Solaris operating system offers no guarantee that the page size request will be honored. You can use `pmap(1)` or `meminfo(2)` to determine page size of the target platform.

**Defaults**

If you specify `-xpagesize=default`, the Solaris operating system sets the page size.

**Expansions**

This option is a macro for `-xpagesize_heap` and `-xpagesize_stack`. These two options accept the same arguments as `-xpagesize`: 4K, 8K, 64K, 512K, 2M, 4M, 32M, 256M, 2G, 16G, or default.

You can set them both with the same value by specifying `-xpagesize` or you can specify them individually with different values.

**Warnings**

The `-xpagesize` option has no effect unless you use it at compile time and at link time. See "3.3.3 Compile-Time and Link-Time Options" on page 50 for a complete list of options that must be specified at both compile time and link time.

**See Also**

Compiling with this option has the same effect as setting the `LD_PRELOAD` environment variable to `mpss.so.1` with the equivalent options, or running the Solaris command `ppgsz(1)` with the equivalent options before running the program. See the Solaris man pages for details.

A.2.160  `-xpagesize_heap=n`

Set the page size in memory for the heap.

A.2.160.1  **Values**

`n` can be 4K, 8K, 64K, 512K, 2M, 4M, 32M, 256M, 2G, 16G, or default. You must specify a valid page size for the target platform. If you do not specify a valid page size, the request is silently ignored at run-time.
Use the `getpagesize(3C)` command on the Solaris operating system to determine the number of bytes in a page. The Solaris operating system offers no guarantee that the page size request will be honored. You can use `pmap(1)` or `meminfo(2)` to determine page size of the target platform.

**Defaults**

If you specify `-xpagesize_heap=default`, the Solaris operating system sets the page size.

**Warnings**

The `-xpagesize_heap` option has no effect unless you use it at compile time and at link time.

**See Also**

Compiling with this option has the same effect as setting the `LD_PRELOAD` environment variable to `mpss.so.1` with the equivalent options, or running the Solaris command `ppgsz(1)` with the equivalent options before running the program. See the Solaris man pages for details.

### A.2.161 `-xpagesize_stack=n`

Set the page size in memory for the stack.

#### A.2.161.1 Values

`n` can be 4k, 8K, 64K, 512K, 2M, 4M, 32M, 256M, 2G, 16G, or `default`. You must specify a valid page size for the target platform. If you do not specify a valid page size, the request is silently ignored at run-time.

Use the `getpagesize(3C)` command on the Solaris operating system to determine the number of bytes in a page. The Solaris operating system offers no guarantee that the page size request will be honored. You can use `pmap(1)` or `meminfo(2)` to determine page size of the target platform.

**Defaults**

If you specify `-xpagesize_stack=default`, the Solaris operating system sets the page size.

**Warnings**

The `-xpagesize_stack` option has no effect unless you use it at compile time and at link time.

**See Also**

Compiling with this option has the same effect as setting the `LD_PRELOAD` environment variable to `mpss.so.1` with the equivalent options, or running the Solaris command `ppgsz(1)` with the equivalent options before running the program. See the Solaris man pages for details.
A.2.162 -xpch=v

This compiler option activates the precompiled-header feature. The precompiled-header feature may reduce compile time for applications whose source files share a common set of include files containing a large amount of source code. The compiler collects information about a sequence of header files from one source file, and then uses that information when recompiling that source file, and when compiling other source files that have the same sequence of headers. The information that the compiler collects is stored in a precompiled-header file.

You can take advantage of this feature through the -xpch and -xpchstop options in combination with the #pragma hdrstop directive.

See Also:
- “A.2.163 -xpchstop=file” on page 331
- “B.2.8 #pragma hdrstop” on page 368

A.2.162.1 Creating a Precompiled-Header File

When you specify -xpch=v, v can be collect:pch_filename or use:pch_filename. The first time you use -xpch, you must specify the collect mode. The compilation command that specifies -xpch=collect must only specify one source file. In the following example, the -xpch option creates a precompiled-header file called myheader.Cpch based on the source file a.cc:

CC -xpch=collect:myheader a.cc

A valid precompiled-header filename always has the suffix .Cpch. When you specify pch_filename, you can add the suffix or let the compiler add it for you. For example, if you specify cc -xpch=collect:foo a.cc, the precompiled-header file is called foo.Cpch.

When you create a precompiled-header file, pick a source file that contains the common sequence of include files across all the source files with which the precompiled-header file is to be used. The common sequence of include files must be identical across these source files.

Remember, only one source filename value is legal in collect mode. For example, CC -xpch=collect:foo bar.cc is valid, whereas CC -xpch=collect:foo bar.cc foobar.cc is invalid because it specifies two source files.

Using A Precompiled-Header File

Specify -xpch=use:pch_filename to use a precompiled-header file. You can specify any number of source files with the same sequence of include files as the source file that was used to create the precompiled-header file. For example, your command in use mode could look like this: CC -xpch=use:foo.Cpch foo.c bar.cc foobar.cc.

You should only use an existing precompiled-header file if the following is true. If any of the following is not true, you should recreate the precompiled-header file:
The compiler that you are using to access the precompiled-header file is the same as the compiler that created the precompiled-header file. A precompiled-header file created by one version of the compiler may not be usable by another version of the compiler, including differences caused by installed patches.

Except for the -xpch option, the compiler options you specify with -xpch=use must match the options that were specified when the precompiled-header file was created.

The set of included headers you specify with -xpch=use is identical to the set of headers that were specified when the precompiled header was created.

The contents of the included headers that you specify with -xpch=use is identical to the contents of the included headers that were specified when the precompiled header was created.

The current directory (that is, the directory in which the compilation is occurring and attempting to use a given precompiled-header file) is the same as the directory in which the precompiled-header file was created.

The initial sequence of pre-processing directives, including #include directives, in the file you specified with -xpch=collect are the same as the sequence of pre-processing directives in the files you specify with -xpch=use.

In order to share a precompiled-header file across multiple source files, those source files must share a common set of include files as their initial sequence of tokens. This initial sequence of tokens is known as the viable prefix. The viable prefix must be interpreted consistently across all the source files that use the same precompiled-header file.

The viable prefix of a source file can only be comprised of comments and any of the following pre-processor directives:

```cpp
#include
#if/ifdef/ifndef/else/elif/endif
#define/undef
#ifident (if identical, passed through as is)
#pragma (if identical)
```

Any of these may reference macros. The #else, #elif, and #endif directives must match within the viable prefix.

Within the viable prefix of each file that shares a precompiled-header file, each corresponding #define and #undef directive must reference the same symbol (in the case of #define, each one must reference the same value). Their order of appearance within each viable prefix must be the same as well. Each corresponding pragma must also be the same and appear in the same order across all the files sharing a precompiled header.

A header file that is incorporated into a precompiled-header file must not violate the following. The results of compiling a program that violate any of these constraints is undefined.

- The header file must not contain function and variable definitions.
The header file must not use `__DATE__` and `__TIME__`. Use of these pre-processor macros can generate unpredictable results.

The header file must not contain `#pragma hdrstop`.

The header file must not use `__LINE__` and `__FILE__` in the viable prefix. It is allowed to use `__LINE__` and `__FILE__` in included headers.

### How to Modify make Files

Here are possible approaches to modifying your make files in order to incorporate `-xpch` into your builds.

- You can use the implicit make rules by using an auxiliary `CCFLAGS` variable and the `KEEP_STATE` facility of both `make` and `dmake`. The precompiled header is produced as a separate, independent step.

```makefile
.KEEP_STATE:
CCFLAGS_AUX = -O etc
CCFLAGS = -xpch=use:shared $(CCFLAGS_AUX)
shared.Cpch: foo.cc
    $(CCC) -xpch=collect:shared $(CCFLAGS_AUX) foo.cc
a.out: foo.o ping.o pong.o
    $(CCC) foo.o ping.o pong.o
```

You can also define your own compilation rule instead of trying to use an auxiliary `CCFLAGS`.

```makefile
.KEEP_STATE:
.SUFFIXES: .o .cc
%.o: %.cc shared.Cpch
    $(CCC) -xpch=use:shared $(CCFLAGS) -c $<
shared.Cpch: foo.cc
    $(CCC) -xpch=collect:shared $(CCFLAGS) foo.cc -xe
a.out: foo.o ping.o pong.o
    $(CCC) foo.o ping.o pong.o
```

- You can produce the precompiled header as a side effect of regular compilation, and without using `KEEP_STATE`, but this approach requires explicit compilation commands.

```makefile
shared.Cpch + foo.o: foo.cc bar.h
    $(CCC) -xpch=collect:shared foo.cc $(CCFLAGS) -c
ping.o: ping.cc shared.Cpch bar.h
    $(CCC) -xpch=use:shared ping.cc $(CCFLAGS) -c
pong.o: pong.cc shared.Cpch bar.h
    $(CCC) -xpch=use:shared pong.cc $(CCFLAGS) -c
a.out: foo.o ping.o pong.o
    $(CCC) foo.o ping.o pong.o
```
A.2.163 \texttt{-xpchstop=\textit{file}}

Use the \texttt{-xpchstop=\textit{file}} option to specify the last include file to be considered in creating the precompile-header file with the \texttt{-xpch} option. Using \texttt{-xpchstop} on the command line is equivalent to placing a \texttt{hdrstop} pragma after the first include-directive that references file in each of the source files that you specify with the \texttt{cc} command.

In the following example, the \texttt{-xpchstop} option specifies that the viable prefix for the precompiled header file ends with the include of \texttt{projectheader.h}. Therefore, \texttt{privateheader.h} is not a part of the viable prefix.

```
example% cat a.cc
#include <stdio.h>
#include <strings.h>
#include "projectheader.h"
#include "privateheader.h"
```

```
example% CC -xpch=collect:foo.Cpch a.cc -xpchstop=projectheader.h -c
```

A.2.163.1 See also
- \texttt{-xpch}, \texttt{pragma hdrstop}

A.2.164 \texttt{-xpec[={yes|no}]}

Generates a Portable Executable Code (PEC) binary. PEC binaries can be used with the Automatic Tuning System (ATS) which works by rebuilding the compiled PEC binary for tuning and troubleshooting - the original source code is not required. More information about the ATS is available at http://cooltools.sunsource.net/ats/index.html.

A binary that is built with \texttt{-xpec} is usually five to ten times larger than if it is built without \texttt{-xpec}.

If you do not specify \texttt{-xpec}, the compiler sets it to \texttt{-xpec=no}. If you specify \texttt{-xpec}, but do not supply a flag, the compiler sets it to \texttt{-xpec=yes}.

A.2.165 \texttt{--xpg}

Compiles for profiling with the \texttt{gprof} profiler.

The \texttt{--xpg} option compiles self-profiling code to collect data for profiling with \texttt{gprof}. This option invokes a runtime recording mechanism that produces a \texttt{gmon.out} file when the program normally terminates.
A.2 Option Reference

---

**Note** – There is no advantage for `-xprofile` if you specify `-xpg`. The two do not prepare or use data provided by the other.

Profiles are generated by using `prof(1)` or `gprof(1)` on 64-bit Solaris platforms or just `gprof` on 32-bit Solaris platforms and include approximate user CPU times. These times are derived from PC sample data (see `pcsample(2)`) for routines in the main executable and routines in shared libraries specified as linker arguments when the executable is linked. Other shared libraries (libraries opened after process startup using `dlopen(3DL)`) are not profiled.

On 32-bit Solaris systems, profiles generated using `prof(1)` are limited to routines in the executable. 32-bit shared libraries can be profiled by linking the executable with `-xpg` and using `gprof(1)`.

The Solaris 10 software does not include system libraries compiled with `-p`. As a result, profiles collected on Solaris 10 platforms do not include call counts for system library routines.

### A.2.165.1 Warnings

If you compile and link separately, and you compile with `-xpg`, be sure to link with `-xpg`. See “3.3.3 Compile-Time and Link-Time Options” on page 50 for a complete list of options that must be specified at both compile time and link time.

See also

```
-xprofile=p, the analyzer(1) man page, and the Performance Analyzer manual.
```

### A.2.166 `-xport64[=(v)]`

Use this option to help you debug code you are porting to a 64-bit environment. Specifically, this option warns against problems such as truncation of types (including pointers), sign extension, and changes to bit-packing that are common when code is ported from a 32-bit architecture such as V8 to a 64-bit architecture such as V9.

#### A.2.166.1 Values

The following table lists the valid values for `v`:
The `-xport64` Values

<table>
<thead>
<tr>
<th>Values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>Generate no warnings related to the porting of code from a 32 bit environment to a 64 bit environment.</td>
</tr>
<tr>
<td>implicit</td>
<td>Generate warning only for implicit conversions. Do not generate warnings when an explicit cast is present.</td>
</tr>
<tr>
<td>full</td>
<td>Generate all warnings related to the porting of code from a 32 bit environment to a 64 bit environment. This includes warnings for truncation of 64-bit values, sign-extension to 64 bits under ISO value-preserving rules, and changes to packing of bitfields.</td>
</tr>
</tbody>
</table>

Defaults

If you do not specify `-xport64`, the default is `-xport64=no`. If you specify `-xport64`, but do not specify a flag, the default is `-xport64=full`.

Examples

This section provides examples of code that can cause truncation of type, sign extension and changes to bit-packing.

Checking for the Truncation of 64-bit Values

When you port to a 64-bit architecture such as V9, your data may be truncated. The truncation could happen implicitly, by assignment, at initialization, or by an explicit cast. The difference of two pointers is the typedef `ptrdiff_t`, which is a 32-bit integer type in 32-bit mode, and a 64-bit integer type in 64-bit mode. The truncation of a long to a smaller size integral type generates a warning as in the following example.

```
example% cat test1.c
int x[10];
int diff = &x[10] - &x[5]; //warn
```

```
example% CC -c -xarch=v9 -Qoption ccfe -xport64=full test1.c
"test1.c", line 3: Warning: Conversion of 64-bit type value to "int" causes truncation.
1 Warning(s) detected.
example%
```

Use `-xport64=implicit` to disable truncation warnings in 64-bit compilation mode when an explicit cast is the cause of data truncation.

```
example% CC -c -xarch=v9 -Qoption ccfe -xport64=implicit test1.c
"test1.c", line 3: Warning: Conversion of 64-bit type value to "int" causes truncation.
```

```
Another common issue that arises from porting to a 64-bit architecture is the truncation of a pointer. This is always an error in C++. An operation such as casting a pointer to an int which causes such a truncation results in an error diagnostic in V9 when you specify -xport64.

```c
example% cat test2.c
char* p;
int main() {
    p = (char*) (((unsigned int)p) & 0xFF); // -xarch=v9 error
    return 0;
}
example% CC -c -xarch=v9 -Qoption ccfe -xport64=full test2.c
"test2.c", line 3: Error: Cannot cast from char* to unsigned.
1 Error(s) detected.
example%
```

**Checking for Sign Extension**

You can also use the -xport64 option to check for situations in which the normal ISO C value-preserving rules allow for the extension of the sign of a signed-integral value in an expression of unsigned-integral type. Such sign extensions can cause subtle run-time bugs.

```c
example% cat test3.c
int i = -1;
void promo(unsigned long l) {}

int main() {
    unsigned long l;
    l = i; // warn
    promo(l); // warn
}
example% CC -c -xarch=v9 -Qoption ccfe -xport64=full test3.c
"test3.c", line 6: Warning: Sign extension from "int" to 64-bit integer.
"test3.c", line 7: Warning: Sign extension from "int" to 64-bit integer.
2 Warning(s) detected.
```

**Checking for Changes to Packing of Bitfields**

Use -xport64 to generate warnings against long bitfields. In the presence of such bitfields, packing of the bitfields might drastically change. Any program which relies on assumptions regarding the way bitfields are packed needs to be reviewed before a successful port can take place to a 64-bit architecture.

```c
example% cat test4.c
#include <stdio.h>
```
union U {
    struct S {
        unsigned long b1:20;
        unsigned long b2:20;
    } s;
    long buf[2];
} u;

int main() {
    u.s.b1 = 0XFFFFF;
    u.s.b2 = 0XFFFFF;
    printf(" u.buf[0] = %lx u.buf[1] = %lx\n", u.buf[0], u.buf[1]);
    return 0;
}

Example:

```
Output in V9:

example% u.buf[0] = ffffffffff00000000 u.buf[1] = 0
```

**Warnings**

Note that warnings are generated only when you compile in 64-bit mode by specifying options such as `-m64`.

**See Also**

“A.2.51 -m32|-m64” on page 256

**A.2.167 -xprefetch[=a[, a...]]**

Enable prefetch instructions on those architectures that support prefetch.

Explicit prefetching should only be used under special circumstances that are supported by measurements.

`a` must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>Enable automatic generation of prefetch instructions</td>
</tr>
</tbody>
</table>
With \texttt{-xprefetch} and \texttt{-xprefetch=auto} the compiler is free to insert prefetch instructions into the code it generates. This may result in a performance improvement on architectures that support prefetch.

If you are running computationally intensive codes on large multiprocessors, you might find it advantageous to use \texttt{-xprefetch=latx:factor}. This option instructs the code generator to adjust the default latency time between a prefetch and its associated load or store by the specified factor.

The prefetch latency is the hardware delay between the execution of a prefetch instruction and the time the data being prefetched is available in the cache. The compiler assumes a prefetch latency value when determining how far apart to place a prefetch instruction and the load or store instruction that uses the prefetched data.

\textbf{Note} – The assumed latency between a prefetch and a load may not be the same as the assumed latency between a prefetch and a store.

The compiler tunes the prefetch mechanism for optimal performance across a wide range of machines and applications. This tuning may not always be optimal. For memory-intensive applications, especially applications intended to run on large multiprocessors, you may be able to obtain better performance by increasing the prefetch latency values. To increase the values, use a factor that is greater than 1 (one). A value between .5 and 2.0 will most likely provide the maximum performance.

For applications with data sets that reside entirely within the external cache, you may be able to obtain better performance by decreasing the prefetch latency values. To decrease the values, use a factor that is less than 1 (one).

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no%auto</td>
<td>Disable automatic generation of prefetch instructions</td>
</tr>
<tr>
<td>explicit</td>
<td>(SPARC) Enable explicit prefetch macros</td>
</tr>
<tr>
<td>no%explicit</td>
<td>(SPARC) Disable explicit prefetch macros</td>
</tr>
<tr>
<td>latx:factor</td>
<td>Adjust the compiler’s assumed prefetch-to-load and prefetch-to-store latencies by the specified factor. You can only combine this flag with \texttt{-xprefetch=auto}. The factor must be a positive floating-point or integer number.</td>
</tr>
<tr>
<td>yes</td>
<td>Obsolete, do not use. Use \texttt{-xprefetch=auto,explicit} instead.</td>
</tr>
<tr>
<td>no</td>
<td>Obsolete, do not use. Use \texttt{-xprefetch=no%auto,no%explicit} instead.</td>
</tr>
</tbody>
</table>
To use the -xprefetch=latx:factor option, start with a factor value near 1.0 and run performance tests against the application. Then increase or decrease the factor, as appropriate, and run the performance tests again. Continue adjusting the factor and running the performance tests until you achieve optimum performance. When you increase or decrease the factor in small steps, you will see no performance difference for a few steps, then a sudden difference, then it will level off again.

### A.2.167.1 Defaults

The default is -xprefetch=auto, explicit. This default adversely affects applications that have essentially non-linear memory access patterns. Specify -xprefetch=no%auto, no%explicit to override the default.

The default of auto is assumed unless explicitly overridden with an argument of no%auto or an argument of no. For example, -xprefetch=explicit is the same as -xprefetch=explicit, auto.

The default of explicit is assumed unless explicitly overridden with an argument of no%explicit or an argument of no. For example, -xprefetch=auto is the same as -xprefetch=auto, explicit.

If only -xprefetch is specified, -xprefetch=auto, explicit is assumed.

If automatic prefetching is enabled, but a latency factor is not specified, then -xprefetch=latx:1.0 is assumed.

### Interactions

This option accumulates instead of overrides.

The sun_prefetch.h header file provides the macros for specifying explicit prefetch instructions. The prefetches will be approximately at the place in the executable that corresponds to where the macros appear.

To use the explicit prefetch instructions, you must be on the correct architecture, include sun_prefetch.h, and either exclude -xprefetch from the compiler command or use -xprefetch, -xprefetch=auto, explicit or -xprefetch=explicit.

If you call the macros and include the sun_prefetch.h header file, but specify -xprefetch=no%explicit, the explicit prefetches will not appear in your executable.

The use of latx:factor is valid only when automatic prefetching is enabled. That is, latx:factor is ignored unless you use it in conjunction with -xprefetch=auto, latx:factor.

### Warnings

Explicit prefetching should only be used under special circumstances that are supported by measurements.
Because the compiler tunes the prefetch mechanism for optimal performance across a wide range of machines and applications, you should only use \(-\text{xprefetch}=%.\text{latx} : \text{factor}\) when the performance tests indicate there is a clear benefit. The assumed prefetch latencies may change from release to release. Therefore, retesting the effect of the latency factor on performance whenever switching to a different release is highly recommended.

**A.2.168 -xprefetch_auto_type=\(a\)**

Where \(a\) is [no\%] indirect_array_access. 

Use this option to determine whether or not the compiler generates indirect prefetches for the loops indicated by the option \(-\text{xprefetch}\_\text{level}\) in the same fashion the prefetches for direct memory accesses are generated.

If you do not specify a setting for \(-\text{xprefetch}\_\text{auto}\_\text{type}\), the compiler sets it to \(-\text{xprefetch}\_\text{auto}\_\text{type}=\text{no%indirect_array_access}\).

Options such as \(-\text{xddepend}\), \(-\text{xrestrict}\), and \(-\text{xalias}\_\text{level}\) can affect the aggressiveness of computing the indirect prefetch candidates and therefore the aggressiveness of the automatic indirect prefetch insertion due to better memory alias disambiguation information.

**A.2.169 -xprefetch_level=[=\(i\)]**

Use the \(-\text{xprefetch}\_\text{level}=\(i\)\) option to control the aggressiveness of the automatic insertion of prefetch instructions as determined with \(-\text{xprefetch}=\text{auto}\). The compiler becomes more aggressive, or in other words, introduces more prefetches, with each higher, level of \(-\text{xprefetch}\_\text{level}\).

The appropriate value for \(-\text{xprefetch}\_\text{level}\) depends on the number of cache misses your application has. Higher \(-\text{xprefetch}\_\text{level}\) values have the potential to improve the performance of applications with a high number of cache misses.

**A.2.169.1 Values**

\(i\) must be one of 1, 2, or 3.

**TABLE A–44** The \(-\text{xprefetch}\_\text{level}\) Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Enables automatic generation of prefetch instructions.</td>
</tr>
</tbody>
</table>
### TABLE A-44  The -xprefetch_level Values (Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Targets additional loops, beyond those targeted at -xprefetch_level=1, for prefetch insertion. Additional prefetches may be inserted beyond those that were inserted at -xprefetch_level=1.</td>
</tr>
<tr>
<td>3</td>
<td>Targets additional loops, beyond those targeted at -xprefetch_level=2, for prefetch insertion. Additional prefetches may be inserted beyond those that were inserted at -xprefetch_level=2.</td>
</tr>
</tbody>
</table>

**Defaults**

The default is -xprefetch_level=1 when you specify -xprefetch=auto.

**Interactions**

This option is effective only when it is compiled with -xprefetch=auto, with optimization level 3 or greater (-xO3), and on a platform that supports prefetch (v9, v9a, v9b, generic64, native64).

### A.2.170 -xprofile=p

Use this option to first collect and save execution-frequency data so that you can then use the data in subsequent runs to improve performance. This option is only valid when you specify optimization at level -xO2 or above.

Compiling with high optimization levels (for example -xO5) is enhanced by providing the compiler with runtime-performance feedback. In order to produce runtime-performance feedback, you must compile with -xprofile=collect, run the executable against a typical data set, and then recompile at the highest optimization level and with -xprofile=use.

Profile collection is safe for multithreaded applications. That is, profiling a program that does its own multitasking (-mt) produces accurate results. This option is only valid when you specify optimization at level -xO2 or above.

### A.2.170.1 Values

p must be one of the following values.

- **collect[:name]**

  Collects and saves execution frequency for later use by the optimizer with -xprofile=use. The compiler generates code to measure statement execution frequency.
The *name* is the optional name of the directory in which profile data will be stored when the program is executed. If specified, *name* should be an absolute UNIX pathname. If *name* is not specified, profile data for a profiled program named *program* will be stored in a directory named *program*.*profile* in the current working directory at the time that *program* is executed.

At runtime, a program compiled with `-xprofile=collect:*name` creates the subdirectory *name*.*profile* to hold the runtime feedback information. Data is written to the file *feedback* in this subdirectory. You can use the `$SUN_PROFDATA` and `$SUN_PROFDATA_DIR` environment variables to change the location of the feedback information. See the *Interactions* section for more information.

If you run the program several times, the execution frequency data accumulates in the *feedback* file; that is, output from prior runs is not lost.

If you are compiling and linking in separate steps, make sure that any object files compiled with `-xprofile=collect` are also linked with `-xprofile=collect`. See “3.3.3 Compile-Time and Link-Time Options” on page 50 for a complete list of options that must be specified at both compile time and link time.

The three options `-xMerge -ztext -xprofile=collect` should not be used together. While `-xMerge` forces statically initialized data into read-only storage, `-ztext` prohibits position-dependent symbol relocations in read-only storage, and `-xprofile=collect` generates statically initialized, position-dependent symbol relocations in writable storage.

- use[:*name*]

The program is optimized by using the executions-frequency data generated and saved in the *feedback* files from a previous execution of the program that was compiled with `-xprofile=collect`.

The *name* is the name of the executable that is being analyzed. The *name* is optional and, if not specified, is assumed to be `a.out`.

Except for the `-xprofile` option which changes from `-xprofile=collect` to `-xprofile=use`, the source files and other compiler options must be exactly the same as those used for the compilation that created the compiled program which in turn generated the *feedback* file. The same version of the compiler must be used for both the collect build and the use build as well. If compiled with `-xprofile=collect:*name`, the same program name *name* must appear in the optimizing compilation: `-xprofile=use:*name`.

The association between an object file and its profile data is based on the UNIX pathname of the object file when it is compiled with `-xprofile=collect`. In some circumstances, the compiler will not associate an object file with its profile data: the object file has no profile data because it was not previously compiled with `-xprofile=collect`, the object file is not linked in a program with `-xprofile=collect`, the program has never been executed.

The compiler can also become confused if an object file was previously compiled in a different directory with `-xprofile=collect` and this object file shares a common basename with other object files compiled with `-xprofile=collect` but they cannot be uniquely
identified by the names of their containing directories. In this case, even if the object file has profile data, the compiler will not be able to find it in the feedback directory when the object file is recompiled with -xprofile=use.

All of these situations cause the compiler to lose the association between an object file and its profile data. Therefore, if an object file has profile data but the compiler is unable to associate it with the object file’s pathname when you specify -xprofile=use, use the -xprofile_pathmap option to identify correct directory. See “A.2.172 -xprofile_pathmap” on page 342.

- tcov

Basic block coverage analysis using the new style tcov.

This option is the new style of basic block profiling for tcov. It has similar functionality to the -xa option, but correctly collects data for programs that have source code in header files or make use of C++ templates. Code instrumentation is similar to that of the -xa option, but .d files are no longer generated. Instead, a single file is generated, the name of which is based on the final executable. For example, if the program is run out of /foo/bar/myprog.profile, then the data file is stored in /foo/bar/myprog.profile/myprog.tcovd.

When running tcov, you must pass it the -x option to force it to use the new style of data. If you do not pass -x, tcov uses the old .d files by default, and produces unexpected output.

Unlike the -xa option, the TCOVDIR environment variable has no effect at compile time. However, its value is used at program runtime.

Interactions

The -xprofile=tcov and the -xa options are compatible in a single executable. That is, you can link a program that contains some files that have been compiled with -xprofile=tcov and other files compiled with -xa. You cannot compile a single file with both options.

The code coverage report produced by -xprofile=tcov can be unreliable if there is inlining of functions due to the use of -xinline or -x04.

You can set the environment variables $SUN_PROFDATA and $SUN_PROFDATA_DIR to control where a program that is compiled with -xprofile=collect puts the profile data. If these variables are not set, the profile data is written to name.profile/feedback in the current directory (name is the name of the executable or the name specified in the -xprofile=collect:name flag). If these variables are set, the -xprofile=collect data is written to $SUN_PROFDATA_DIR/$SUN_PROFDATA.

The $SUN_PROFDATA and $SUN_PROFDATA_DIR environment variables similarly control the path and names of the profile data files written by tcov. See the tcov(1) man page for more information.
Warnings

If you compile and link in separate steps, the same `-xprofile` option must appear in both the compile command and the link command. Including `-xprofile` in one step and excluding it from the other step will not affect the correctness of the program, but you will not be able to do profiling.

`-xprofile=collect` or `-xprofile=tcov` should not be used with `-G` to build shared libraries on Linux platforms.

See also

`-xa, tcov(1)` man page, Program Performance Analysis Tools.

A.2.171 `-xprofile_ircache[=path]`

(SPARC) Use `-xprofile_ircache[=path]` with `-xprofile=collect` to use to improve compilation time during the use phase by reusing compilation data saved from the collect phase.

With large programs, compilation time in the use phase can improve significantly because the intermediate data is saved. Note that the saved data could increase disk space requirements considerably.

When you use `-xprofile_ircache[=path]`, `path` overrides the location where the cached files are saved. By default, these files are saved in the same directory as the object file. Specifying a path is useful when the collect and use phases happen in two different directories. Here’s a typical sequence of commands:

```
example% CC -xO5 -xprofile=collect -xprofile_ircache t1.cc t2.cc
example% a.out    // run collects feedback data
example% CC -xO5 -xprofile=use -xprofile_ircache t1.cc t2.cc
```

A.2.172 `-xprofile_pathmap`

(SPARC) Use the `-xprofile_pathmap=collect_prefix:use_prefix` option when you are also specifying the `-xprofile=use` command. Use `-xprofile_pathmap` when both of the following are true and the compiler is unable to find profile data for an object file that is compiled with `-xprofile=use`.

- You are compiling the object file with `-xprofile=use` in a directory that is different from the directory in which the object file was previously compiled with `-xprofile=collect`.
- Your object files share a common basename in the profile but are distinguished from each other by their location in different directories.
The `collect-prefix` is the prefix of the UNIX pathname of a directory tree in which object files were compiled using `-xprofile=collect`.

The `use-prefix` is the prefix of the UNIX pathname of a directory tree in which object files are to be compiled using `-xprofile=use`.

If you specify multiple instances of `-xprofile_pathmap`, the compiler processes them in the order of their occurrence. Each `use-prefix` specified by an instance of `-xprofile_pathmap` is compared with the object file pathname until either a matching `use-prefix` is identified or the last specified `use-prefix` is found not to match the object file pathname.

### A.2.173 `-xreduction`

Analyzes loops for reduction in automatic parallelization. This option is valid only if `-xautopar` is also specified. Otherwise the compiler issues a warning.

When reduction recognition is enabled, the compiler parallelizes reductions such as dot products, maximum and minimum finding. These reductions yield different roundoffs from those obtained by unparallelized code.

### A.2.174 `-xregs=r[, r...]`

Controls scratch register usage.

The compiler can generate faster code if it has more registers available for temporary storage (scratch registers). This option makes available additional scratch registers that might not always be appropriate.

#### A.2.174.1 Values

`r` must be one of the following values. The meaning of each value depends upon the `-m32` or `-m64` setting.
TABLE A–45  The -xregs Values

<table>
<thead>
<tr>
<th>Values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]appl</td>
<td>(SPARC) [Does not] Allow the compiler to generate code using the application registers as scratch registers. The application registers are: g2, g3, g4 (on 32–bit platforms) g2, g3 (on 64–bit platforms) It is strongly recommended that all system software and libraries be compiled using -xregs=no%appl. System software (including shared libraries) must preserve these registers’ values for the application. Their use is intended to be controlled by the compilation system and must be consistent throughout the application. In the SPARC ABI, these registers are described as application registers. Using these registers can increase performance because fewer load and store instructions are needed. However, such use can conflict with programs that use the registers for other purposes.</td>
</tr>
<tr>
<td>[no%]float</td>
<td>(SPARC) [Does not] Allow the compiler to generate code by using the floating-point registers as scratch registers for integer values. Use of floating-point values may use these registers regardless of this option. If you want your code to be free of all references to floating point registers, you need to use -xregs=no%float and also make sure your code does not use floating point types in any way.</td>
</tr>
</tbody>
</table>
### TABLE A-45  The -xregs Values  (Continued)

<table>
<thead>
<tr>
<th>Values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>([no%]frameptr) (x86)</td>
<td>[Does not] Allow the compiler to use the frame-pointer register (%ebp on IA32, %rbp on AMD64) as an unallocated callee-saves register.</td>
</tr>
</tbody>
</table>

Using this register as an unallocated callee-saves register may improve program run time. However, it also reduces the capacity of some tools to inspect and follow the stack. This stack inspection capability is important for system performance measurement and tuning. Therefore, using this optimization may improve local program performance at the expense of global system performance.

- Tools, such as the Performance Analyzer, that dump the stack for postmortem diagnosis will not work.
- Debuggers (adb, mdb, dbx) will not be able to dump the stack or directly pop stack frames.
- The dtrace performance analysis facility will be unable to collect information on any frames on the stack before the most recent frame missing the frame pointer.
- Posix pthread_cancel will fail trying to find cleanup handlers.
- C++ exceptions cannot propagate through C functions.
  The failures in C++ exceptions occur when a C function that has lost its frame pointer calls a C++ function that throws an exception through the C function. Such calls typically occur when a function accepts a function pointer (for example, qsort) or when a global function, such as malloc, is interposed upon.
  The last two affects listed above may impact the correct operation of applications. Most application code will not encounter these problems.
  Libraries that are developed by using -x04, however, need documentation that details the restrictions of their usage by their clients.
- Note - The compiler ignores -xregs=framptr and issues a warning if you also specify -xpg. Also, the compiler ignores this option for 32-bit x86 compilation unless exceptions are disabled with -noex option.

### Defaults

The SPARC default is -xregs=appl,float.

The x86 default is -xregs=no%frameptr -xregs=frameptr in included in the expansion of -fast.

### Examples

To compile an application program using all available scratch registers, use -xregs=appl,float.
To compile non-floating-point code that is sensitive to context switch, use
-xregs=no%appl,no%float.

See also
SPARC V8 and SPARC V9 ABI

A.2.175  -xrestrict=[=f]

(SPARC) Treats pointer-valued function parameters as restricted pointers. f must be one of the
following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%all</td>
<td>All pointer parameters in the entire file are treated as restricted.</td>
</tr>
<tr>
<td>%none</td>
<td>No pointer parameters in the file are treated as restricted.</td>
</tr>
<tr>
<td>%source</td>
<td>Only functions defined within the main source file are restricted. Functions defined within included files are not restricted.</td>
</tr>
<tr>
<td>fn[fn...]</td>
<td>A comma-separated list of one or more function names. If you specify a function list, the compiler treats pointer parameters in the specified functions as restricted; Refer to the following section, “A.2.175.1 Restricted Pointers” on page 346, for more information.</td>
</tr>
</tbody>
</table>

This command-line option can be used on its own, but it is best used with optimization. For example, the command:

```
%CC -xO3 -xrestrict=%all prog.cc
```
treats all pointer parameters in the file prog.c as restricted pointers. The command:

```
%CC -xO3 -xrestrict=agc prog.cc
```
treats all pointer parameters in the function agc in the file prog.c as restricted pointers.

The default is %none; specifying -xrestrict is equivalent to specifying -xrestrict=%source.

A.2.175.1  Restricted Pointers

In order for a compiler to effectively perform parallel execution of a loop, it needs to determine if certain lvalues designate distinct regions of storage. Aliases are lvalues whose regions of storage are not distinct. Determining if two pointers to objects are aliases is a difficult and time consuming process because it could require analysis of the entire program. Consider function vsq() below:
EXAMPLE A–3  A Loop With Two Pointers

```c
extern "C"
void vsq(int n, double *a, double *b) {
    int i;
    for (i=0; i<n; i++) {
        b[i] = a[i] * a[i];
    }
}
```

The compiler can parallelize the execution of the different iterations of the loops if it knows that pointers `a` and `b` access different objects. If there is an overlap in objects accessed through pointers `a` and `b` then it would be unsafe for the compiler to execute the loops in parallel.

At compile time, the compiler does not know if the objects accessed by `a` and `b` overlap by simply analyzing the function `vsq()`. the compiler may need to analyze the whole program to get this information. You can specify that pointer-valued function parameters be treated as restricted pointers by using the following command line option: `-xrestrict=[func1,...,funcn]` If a function list is specified, pointer parameters in the specified functions are treated as restricted; otherwise, all pointer parameters in the entire source file are treated as restricted (not recommended). For example, `-xrestrict=vsq` qualifies the pointers `a` and `b` given in the example of the function `vsq()`.

Declaring the pointer arguments as restricted states that the pointers designate distinct objects. The compiler can assume that `a` and `b` point to distinct regions of storage. With this alias information, the compiler is able to parallelize the loop.

It is critical that you use `-xrestrict` correctly. If pointers qualified as restricted pointers point to objects that are not distinct, the compiler can incorrectly parallelize loops resulting in undefined behavior. For example, assume that pointers `a` and `b` of function `vsq()` point to objects that overlap, such that `b[i]` and `a[i+1]` are the same object. If `a` and `b` are not declared as restricted pointers the loops will be executed serially. If `a` and `b` are incorrectly qualified as restricted pointers the compiler may parallelize the execution of the loops, which is not safe, because `b[i+1]` should only be computed after `b[i]` is computed.

A.2.176  -xS

Allows debugging by `dbx` without object (.o) files.

This option causes all the debug information to be copied into the executable. This has little impact on dbx performance or the run-time performance of the program, but it does take more disk space.

This option has an effect only with `-xdebugformat=stabs`, where the default is not to copy debug data into the executable. With the default debug format `-xdebugformat=dwarf`, debug data is always copied into the executable, and there is no option to prevent the copying.
A.2.177  \(-x\text{safe}=\text{mem}\)

SPARC: Allows the compiler to assume that no memory protection violations occur.

This option allows the compiler to use the non-faulting load instruction in the SPARC V9 architecture.

A.2.177.1  Interactions

This option takes effect only when used with optimization level \(-xO5\) and one of the following \(-\text{arch}\) values: \(\text{sparc}\), \(\text{sparcvis}\), or \(\text{sparcvis2}\) for both \(-m32\) and \(-m64\).

A.2.177.2  Warnings

Because non-faulting loads do not cause a trap when a fault such as address misalignment or segmentation violation occurs, you should use this option only for programs in which such faults cannot occur. Because few programs incur memory-based traps, you can safely use this option for most programs. Do not use this option for programs that explicitly depend on memory-based traps to handle exceptional conditions.

A.2.178  \(-xs\)

Deprecated - do not use. The source browser functionality is obsolete. This options is silently ignored.

A.2.179  \(-xs\text{fast}\)

Deprecated - do not use. The source browser functionality is obsolete. This option is silently ignored.

A.2.180  \(-x\text{space}\)

SPARC: Does not allow optimizations that increase code size.

A.2.181  \(-x\text{target}=t\)

Specifies the target platform for instruction set and optimization.

The performance of some programs can benefit by providing the compiler with an accurate description of the target computer hardware. When program performance is critical, the proper specification of the target hardware could be very important. This is especially true when running on the newer SPARC processors. However, for most programs and older SPARC processors, the performance gain is negligible and a generic specification is sufficient.
The value of \( t \) must be one of the following: native, generic, native64, generic64, system-name.

Each specific value for -xtarget expands into a specific set of values for the -xarch, -xchip, and -xcache options. Use the -dryrun option to determine the expansion of -xtarget=native on a running system.

For example, -xtarget=ultraT2 is equivalent to: -xarch=sparcvis2 -xchip=ultraT2 -xcache=8/16/4:4096/64/16.

Note – The expansion of -xtarget for a specific host platform might not expand to the same -xarch, -xchip, or -xcache settings as -xtarget=native when compiling on that platform.

### A.2.181.1 -xtarget Values By Platform

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>native</td>
<td>Gets the best performance on the host system. The compiler generates code optimized for the host system. It determines the available architecture, chip, and cache properties of the machine on which the compiler is running.</td>
</tr>
<tr>
<td>native64</td>
<td>Gets the best performance for 64-bit object binaries on the host system. The compiler generates 64-bit object binaries optimized for the host system. It determines the available 64-bit architecture, chip, and cache properties of the machine on which the compiler is running.</td>
</tr>
<tr>
<td>generic</td>
<td>This is the default value. Gets the best performance for generic architecture, chip, and cache.</td>
</tr>
<tr>
<td>generic64</td>
<td>Sets the parameters for the best performance of 64-bit object binaries over most 64-bit platform architectures.</td>
</tr>
<tr>
<td>system-name</td>
<td>Gets the best performance for the specified platform. Select a system name from the following lists for which represents the actual system you are targeting.</td>
</tr>
</tbody>
</table>

### -xtarget Values on SPARC Platforms

Compiling for 64-bit Solaris software on SPARC or UltraSPARC V9 is indicated by the -m64 option. If you specify -xtarget with a flag other than native64 or generic64, you must also specify the -m64 option as follows: -xtarget=ultra... -m64 otherwise the compiler uses a 32-bit memory model.
<table>
<thead>
<tr>
<th><code>-xtarget=</code></th>
<th><code>-xarch</code></th>
<th><code>-xchip</code></th>
<th><code>-xcache</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>generic</td>
<td>generic</td>
<td>generic</td>
<td>generic</td>
</tr>
<tr>
<td>cs6400</td>
<td>v8plusa</td>
<td>super</td>
<td>16/32/4:2048/64/1</td>
</tr>
<tr>
<td>entr150</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>entr2</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>entr2/1170</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>entr2/1200</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>entr2/2170</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>entr2/2200</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>entr3000</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>entr4000</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>entr5000</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>entr6000</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>sc2000</td>
<td>v8plusa</td>
<td>super</td>
<td>16/32/4:2048/64/1</td>
</tr>
<tr>
<td>solb5</td>
<td>v7</td>
<td>old</td>
<td>128/32/1</td>
</tr>
<tr>
<td>solb6</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss1</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>ss10</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/20</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/30</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/40</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/402</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/41</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/412</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/50</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/51</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/512</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/514</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>-xtarget=</td>
<td>-xarch</td>
<td>-xchip</td>
<td>-xcache</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>--------</td>
<td>-----------</td>
</tr>
<tr>
<td>ss10/61</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/612</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/71</td>
<td>v8</td>
<td>super2</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/712</td>
<td>v8</td>
<td>super2</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/hs11</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss10/hs12</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss10/hs14</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss10/hs21</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss10/hs22</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss1000</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss1plus</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>ss2</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ss20</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/151</td>
<td>v8</td>
<td>hyper</td>
<td>512/64/1</td>
</tr>
<tr>
<td>ss20/152</td>
<td>v8</td>
<td>hyper</td>
<td>512/64/1</td>
</tr>
<tr>
<td>ss20/50</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss20/502</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss20/51</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/512</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/514</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/61</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/612</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/71</td>
<td>v8</td>
<td>super2</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/712</td>
<td>v8</td>
<td>super2</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/hs11</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss20/hs12</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss20/hs14</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss20/hs21</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
</tbody>
</table>
### TABLE A–48 -xtarget Expansions on SPARC Architecture (Continued)

<table>
<thead>
<tr>
<th>-xtarget</th>
<th>-xarch</th>
<th>-xchip</th>
<th>-xcache</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss20/hs22</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss2p</td>
<td>v7</td>
<td>powerup</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ss4</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss4/110</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss4/85</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss5</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss5/110</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss5/85</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss600/120</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ss600/140</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ss600/41</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/412</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/51</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/512</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/514</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/61</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/612</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ssiloc</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ssipc</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>ssipx</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>sslc</td>
<td>v8a</td>
<td>micro</td>
<td>2/16/1</td>
</tr>
<tr>
<td>ssllt</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ssllx</td>
<td>v8a</td>
<td>micro</td>
<td>2/16/1</td>
</tr>
<tr>
<td>sslx2</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ssslc</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>sslyger</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>sun4/110</td>
<td>v7</td>
<td>old</td>
<td>2/16/1</td>
</tr>
<tr>
<td>sun4/15</td>
<td>v8a</td>
<td>micro</td>
<td>2/16/1</td>
</tr>
<tr>
<td>Target</td>
<td>xarch</td>
<td>xchip</td>
<td>xcache</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>sun4/150</td>
<td>v7</td>
<td>old</td>
<td>2/16/1</td>
</tr>
<tr>
<td>sun4/20</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>sun4/25</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>sun4/260</td>
<td>v7</td>
<td>old</td>
<td>128/16/1</td>
</tr>
<tr>
<td>sun4/280</td>
<td>v7</td>
<td>old</td>
<td>128/16/1</td>
</tr>
<tr>
<td>sun4/30</td>
<td>v8a</td>
<td>micro</td>
<td>2/16/1</td>
</tr>
<tr>
<td>sun4/330</td>
<td>v7</td>
<td>old</td>
<td>128/16/1</td>
</tr>
<tr>
<td>sun4/370</td>
<td>v7</td>
<td>old</td>
<td>128/16/1</td>
</tr>
<tr>
<td>sun4/390</td>
<td>v7</td>
<td>old</td>
<td>128/16/1</td>
</tr>
<tr>
<td>sun4/40</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>sun4/470</td>
<td>v7</td>
<td>old</td>
<td>128/32/1</td>
</tr>
<tr>
<td>sun4/490</td>
<td>v7</td>
<td>old</td>
<td>128/32/1</td>
</tr>
<tr>
<td>sun4/50</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>sun4/60</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>sun4/630</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>sun4/65</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>sun4/670</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>sun4/690</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>sun4/75</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ultra</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra1/140</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra1/170</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra1/200</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra2</td>
<td>v8plusa</td>
<td>ultra2</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra2/1170</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra2/1200</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:1024/64/1</td>
</tr>
<tr>
<td>ultra2/1300</td>
<td>v8plusa</td>
<td>ultra2</td>
<td>16/32/1:2048/64/1</td>
</tr>
<tr>
<td>ultra2/2170</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
</tbody>
</table>
TABLE A–48  -xtarget Expansions on SPARC Architecture  (Continued)

<table>
<thead>
<tr>
<th>-xtarget</th>
<th>-xarch</th>
<th>-xchip</th>
<th>-xcache</th>
</tr>
</thead>
<tbody>
<tr>
<td>ultra2/2200</td>
<td>v8plusa</td>
<td>ultra</td>
<td>16/32/1:1024/64/1</td>
</tr>
<tr>
<td>ultra2/2300</td>
<td>v8plusa</td>
<td>ultra2</td>
<td>16/32/1:2048/64/1</td>
</tr>
<tr>
<td>ultra2e</td>
<td>v8plusa</td>
<td>ultra2e</td>
<td>16/32/1:256/64/4</td>
</tr>
<tr>
<td>ultra2i</td>
<td>v8plusa</td>
<td>ultra2i</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra3</td>
<td>v8plusa</td>
<td>ultra3</td>
<td>64/32/4:8192/512/1</td>
</tr>
<tr>
<td>ultra3cu</td>
<td>v8plusa</td>
<td>ultra3cu</td>
<td>64/32/4:8192/512/2</td>
</tr>
<tr>
<td>ultra3i</td>
<td>v8plusa</td>
<td>ultra3i</td>
<td>64/32/4:1024/64/4</td>
</tr>
<tr>
<td>ultra4</td>
<td>v8plusa</td>
<td>ultra4</td>
<td>64/32/4:8192/128/2</td>
</tr>
<tr>
<td>ultra4plus</td>
<td>v8plusa</td>
<td>ultra4plus</td>
<td>64/32/4:2048/64/4:32768/64/4</td>
</tr>
<tr>
<td>ultraT1</td>
<td>v8plusa</td>
<td>ultraT1</td>
<td>8/16/4:4:3072/64/12/32</td>
</tr>
<tr>
<td>ultraT2</td>
<td>sparcvis2</td>
<td>ultraT2</td>
<td>8/16/4:4096/64/16</td>
</tr>
<tr>
<td>ultraT2plus</td>
<td>sparcvis2</td>
<td>ultraT2plus</td>
<td>8/16/4:4096/64/16</td>
</tr>
<tr>
<td>sparc64vi</td>
<td>sparcfmaf</td>
<td>sparc64vi</td>
<td>128/64/2:5120/64/10</td>
</tr>
<tr>
<td>sparc64vii</td>
<td>sparcima</td>
<td>sparc64vii</td>
<td>64/64/2:5120/256/10</td>
</tr>
</tbody>
</table>

See “A.2.114-xcache= c” on page 287 for more information on the cache properties of the UltraSPARC IVplus, UltraSPARCT1, and UltraSPARCT2 chips.

-xtarget Values on x86 Platforms

Compiling for 64-bit Solaris software on 64-bit x86 platforms is indicated by the -m64 option. If you specify -xtarget with a flag other than native64 or generic64, you must also specify the -m64 option as follows: -xtarget=opteron ... -m64 otherwise the compiler uses a 32-bit memory model.

TABLE A–49  -xtarget Values on x86 Platforms

<table>
<thead>
<tr>
<th>-xtarget</th>
<th>-xarch</th>
<th>-xchip</th>
<th>-xcache</th>
</tr>
</thead>
<tbody>
<tr>
<td>generic</td>
<td>generic</td>
<td>generic</td>
<td>generic</td>
</tr>
<tr>
<td>opteron</td>
<td>sse2</td>
<td>opteron</td>
<td>64/64/2:1024/64/16</td>
</tr>
<tr>
<td>pentium</td>
<td>386</td>
<td>pentium</td>
<td>generic</td>
</tr>
<tr>
<td>pentium_pro</td>
<td>pentium_pro</td>
<td>pentium_pro</td>
<td>generic</td>
</tr>
<tr>
<td>pentium3</td>
<td>sse</td>
<td>pentium3</td>
<td>16/32/4:256/32/4</td>
</tr>
</tbody>
</table>
TABLE A-49  -xtarget Values on x86 Platforms  (Continued)

<table>
<thead>
<tr>
<th>-xtarget=</th>
<th>-xarch</th>
<th>-xchip</th>
<th>-xcache</th>
</tr>
</thead>
<tbody>
<tr>
<td>pentium4</td>
<td>sse2</td>
<td>pentium4</td>
<td>8/64/4:256/128/8</td>
</tr>
<tr>
<td>nehalem</td>
<td>sse4_2</td>
<td>nehalem</td>
<td>32/64/8:256/64/8:8192/64/16</td>
</tr>
<tr>
<td>penryn</td>
<td>sse4_1</td>
<td>penryn</td>
<td>2/64/8:4096/64/16</td>
</tr>
<tr>
<td>woodcrest</td>
<td>ssse3</td>
<td>core2</td>
<td>32/64/8:4096/64/16</td>
</tr>
<tr>
<td>barcelona</td>
<td>amdsse4a</td>
<td>amdfam10</td>
<td>64/64/2:512/64/16</td>
</tr>
</tbody>
</table>

**Defaults**

On both SPARC and x86 devices, if -xtarget is not specified, -xtarget=generic is assumed.

**Expansions**

The -xtarget option is a macro that permits a quick and easy specification of the -xarch, -xchip, and -xcache combinations that occur on commercially purchased platforms. The only meaning of -xtarget is in its expansion.

**Examples**

-xtarget=sun4/15 means -xarch=v8a -xchip=micro -xcache=2/16/1.

**Interactions**

Compilation for SPARC V9 architecture indicated by the -xarch=v9|v9a|v9b option. Setting -xtarget=ultra or ultra2 is not necessary or sufficient. If -xtarget is specified, the -xarch=v9, v9a, or v9b option must appear after the -xtarget. For example:

-xarch=v9  -xtarget=ultra

expands to the following and reverts the -xarch value to v8.

-xarch=v9  -xarch=v8  -xchip=ultra  -xcache=16/32/1:512/64/1

The correct method is to specify -xarch after -xtarget. For example:

-xtarget=ultra  -xarch=v9

**Warnings**

When you compile and link in separate steps, you must use the same -xtarget settings in the compile step and the link step.
A.2.182 -xthreadvar[=o]

Specify -xthreadvar to control the implementation of thread local variables. Use this option in conjunction with the __thread declaration specifier to take advantage of the compiler’s thread-local storage facility. After you declare the thread variables with the __thread specifier, specify -xthreadvar to enable the use of thread-local storage with position dependent code (non-PIC code) in dynamic (shared) libraries. For more information on how to use __thread, see "4.2 Thread-Local Storage" on page 65.

A.2.182.1 Values

o must be one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]dynamic</td>
<td>[Do not] Compile variables for dynamic loading. Access to thread variables is significantly faster when -xthreadvar=no%dynamic but you cannot use the object file within a dynamic library. That is, you can only use the object file in an executable file.</td>
</tr>
</tbody>
</table>

Defaults

If you do not specify -xthreadvar, the default used by the compiler depends upon whether or not position-independent code is enabled. If position-independent code is enabled, the option is set to -xthreadvar=dynamic. If position-independent code is disabled, the option is set to -xthreadvar=no%dynamic.

If you specify -xthreadvar but do not specify any arguments, the option is set to -xthreadvar=dynamic.

Interactions

the -mt option must be used when compiling and linking files that use __thread.

Warnings

If there is non-position-independent code within a dynamic library, you must specify -xthreadvar.

The linker cannot support the thread-variable equivalent of non-PIC code in dynamic libraries. Non-PIC thread variables are significantly faster, and hence should be the default for executables.

See Also

- xcode, -KPIC, -Kpic
A.2.183  -xtime
Causes the CC driver to report execution time for the various compilation passes.

A.2.184  -xtrigraphs[={yes|no}]
Enables or disables recognition of trigraph sequences as defined by the ISO/ANSI C standard.
If your source code has a literal string containing question marks (?) that the compiler is
interpreting as a trigraph sequence, you can use the -xtrigraph=no suboption to turn off the
recognition of trigraph sequences.

A.2.184.1  Values
You can specify one of the following two values for -xtrigraphs:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Enables recognition of trigraph sequences throughout the compilation unit</td>
</tr>
<tr>
<td>no</td>
<td>Disables recognition of trigraph sequences throughout the compilation unit</td>
</tr>
</tbody>
</table>

Defaults
When you do not include the -xtrigraphs option on the command line, the compiler assumes
-xtrigraphs=yes.
If only -xtrigraphs is specified, the compiler assumes -xtrigraphs=yes.

Examples
Consider the following example source file named trigraphs_demo.cc.

```
#include <stdio.h>

int main ()
{
    (void) printf("(?\?) in a string appears as (??)\n");
    return 0;
}
```

Here is the output if you compile this code with -xtrigraphs=yes.

```
example% CC -xtrigraphs=yes trigraphs_demo.cc
example% a.out
(??) in a string appears as (??)
```
Here is the output if you compile this code with -xtrigraphs=no.

```bash
example% CC -xtrigraphs=no trigraphs_demo.cc
example% a.out
(??) in a string appears as (??)
```

See also

For information on trigraphs, see the C User’s Guide chapter about transitioning to ANSI/ISO C.

A.2.185 -xunroll=n

Enables unrolling of loops where possible.

This option specifies whether or not the compiler optimizes (unrolls) loops.

A.2.185.1 Values

When \( n \) is 1, it is a suggestion to the compiler to not unroll loops.

When \( n \) is an integer greater than 1, \(-\text{unroll}=n\) causes the compiler to unroll loops \( n \) times.

A.2.186 -xustr=\{ascii_utf16_ushort|no\}

Use this option if your code contains string or character literals that you want the compiler to convert to UTF-16 strings in the object file. Without this option, the compiler neither produces nor recognizes sixteen-bit character string literals. This option enables recognition of the U"ASCII_string" string literals as an array of unsigned short int. Since such strings are not yet part of any standard, this option enables recognition of non-standard C++.

Not all files have to be compiled with this option.

A.2.186.1 Values

Specify \(-\text{xustr}=\text{ascii_utf16_ushort}\) if you need to support an internationalized application that uses ISO10646 UTF-16 string literals. You can turn off compiler recognition of U"ASCII_string" string or character literals by specifying \(-\text{xustr}=\text{no}\). The right-most instance of this option on the command line overrides all previous instances.

You can specify \(-\text{xustr}=\text{ascii_ustf16_ushort}\) without also specifying a U"ASCII_string" string literal. It is not an error to do so.
Defaults

The default is -xustr=no. If you specify -xustr without an argument, the compiler won’t accept it and instead issues a warning. The default can change if the C or C++ standards define a meaning for the syntax.

Example

The following example shows a string literal in quotes that is prepended by U. It also shows a command line that specifies -xustr:

```
example% cat file.cc
const unsigned short *foo = U"foo";
const unsigned short bar[] = U"bar";
const unsigned short *fun() {return foo;}
example% CC -xustr=ascii_utf16_ushort file.cc -c
```

An 8-bit character literal can be prepended with U to form a 16-bit UTF-16 character of type unsigned short. Examples:

```
const unsigned short x = U'x';
const unsigned short y = U'\x79';
```

A.2.187 -xvector[=a]

Enables automatic generation of calls to the vector library functions and/or the generation of the SIMD (Single Instruction Multiple Data) instructions. You must use default rounding mode by specifying -fround=nearest when you use this option.

a is the equivalent of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[no%]lib</td>
<td>(Solaris only) Does [not] enable the compiler to transform math library calls within loops into single calls to the equivalent vector math routines when such transformations are possible. This could result in a performance improvement for loops with large loop counts.</td>
</tr>
</tbody>
</table>
### A.2.187 The \texttt{-xvector} Flags (Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{[not]simd}</td>
<td>Does [not] direct the compiler to use the native x86 SSE SIMD instructions to improve performance of certain loops. The compiler can only accept this switch if the target architecture supports SIMD instructions. For example, you must specify \texttt{-xarch=amd64}, \texttt{-xarch=amd64a} or \texttt{-xarch=generic64}. You must also specify an optimization level of \texttt{-xO3} or above as well as \texttt{-xdepend} when you specify \texttt{-xvector=simd}.</td>
</tr>
<tr>
<td>\texttt{yes}</td>
<td>This option is deprecated, specify \texttt{-xvector=lib} instead.</td>
</tr>
<tr>
<td>\texttt{no}</td>
<td>This option is deprecated, specify \texttt{-xvector=none} instead.</td>
</tr>
</tbody>
</table>

#### A.2.187.1 Defaults

The default is \texttt{-xvector=none}. If you specify \texttt{-xvector}, but do not provide a flag, the compiler assumes \texttt{-xvector=lib}.

#### Interactions

If you use \texttt{-xvector} on the command line without previously specifying \texttt{-xdepend}, \texttt{-xvector} triggers \texttt{-xdepend}. The \texttt{-xvector} option also raises the optimization level to \texttt{-xO3} if optimization is not specified or optimization is set lower than \texttt{-xO3}.

The compiler includes the \texttt{libmvec} libraries in the load step.

If you compile and link with separate commands, be sure to use \texttt{-xvector} in the linking \texttt{CC} command. See "3.3.3 Compile-Time and Link-Time Options" on page 50 for a complete list of options that must be specified at both compile time and link time.

### A.2.188 \texttt{-xvis=[\{yes|no\}]} (SPARC)

Use the \texttt{-xvis=[\{yes|no\}]} command when you are using the assembly-language templates defined in the VIS\textsuperscript{TM} instruction-set Software Developers Kit (VSDK).

The VIS instruction set is an extension to the SPARC v9 instruction set. Even though the UltraSPARC processors are 64-bit, there are many cases, especially in multimedia applications, when the data are limited to eight or 16 bits in size. The VIS instructions can process four 16-bit data with one instruction so they greatly improve the performance of applications that handle new media such as imaging, linear algebra, signal processing, audio, video and networking.

#### A.2.188.1 Defaults

The default is \texttt{-xvis=no}. Specifying \texttt{-xvis} is equivalent to specifying \texttt{-xvis=yes}. 
See Also
For more information on the VSDK, see http://www.sun.comprocessors/vis/.

A.2.189  -xvpara
Issues warnings about potential parallel-programming related problems that may cause incorrect results when using OpenMP. Use with -xopenmp and OpenMP API directives.

The compiler issues warnings when it detects the following situations:
- Loops are parallelized using MP directives with data dependencies between different loop iterations
- OpenMP data-sharing attributes-clauses are problematic. For example, declaring a variable "shared" whose accesses in an OpenMP parallel region may cause a data race, or declaring a variable "private" whose value in a parallel region is used after the parallel region.

No warnings appear if all parallelization directives are processed without problems.

Example:
CC -xopenmp -xvpara any.cc

Note – Sun Studio compilers support OpenMP 2.5 API parallelization. Consequently, the MP pragmas directives are deprecated and are no longer supported. See the OpenMP API User’s Guide for information on migrating to the OpenMP API.

A.2.190  -xwe
Converts all warnings to errors by returning nonzero exit status.

A.2.190.1 See Also
"A.2.17 -errwarn[=t]" on page 224

A.2.191  -yc, path
Specifies a new path for the location of component c.
If the location of a component is specified, then the new path name for the component is path/component_name. This option is passed to ld.

A.2.191.1 Values
c must be one of the following values:
TABLE A-53  The -Y Flags

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Changes the default directory for cpp.</td>
</tr>
<tr>
<td>0</td>
<td>Changes the default directory for ccfe.</td>
</tr>
<tr>
<td>a</td>
<td>Changes the default directory for fbe.</td>
</tr>
<tr>
<td>2 (SPARC)</td>
<td>Changes the default directory for iropt.</td>
</tr>
<tr>
<td>c (SPARC)</td>
<td>Changes the default directory for cg.</td>
</tr>
<tr>
<td>O (SPARC)</td>
<td>Changes the default directory for ipo.</td>
</tr>
<tr>
<td>k</td>
<td>Changes the default directory for CCl ink.</td>
</tr>
<tr>
<td>l</td>
<td>Changes the default directory for ld.</td>
</tr>
<tr>
<td>f</td>
<td>Changes the default directory for c++filt.</td>
</tr>
<tr>
<td>m</td>
<td>Changes the default directory for mcs.</td>
</tr>
<tr>
<td>u (x86)</td>
<td>Changes the default directory for ube.</td>
</tr>
<tr>
<td>h (x86)</td>
<td>Changes the default directory for ir2hf.</td>
</tr>
<tr>
<td>A</td>
<td>Specifies a directory to search for all compiler components. If a component is not found in path, the search reverts to the directory where the compiler is installed.</td>
</tr>
<tr>
<td>P</td>
<td>Adds path to the default library search path. This path will be searched before the default library search paths.</td>
</tr>
<tr>
<td>S</td>
<td>Changes the default directory for startup object files</td>
</tr>
</tbody>
</table>

Interactions

You can have multiple -Y options on a command line. If more than one -Y option is applied to any one component, then the last occurrence holds.

See also

Solaris Linker and Libraries Guide

A.2.192  -z[ ]arg

Link editor option. For more information, see the \ld(1) man page and the Solaris Linker and Libraries Guide.
This appendix describes the C++ compiler pragmas. A *pragma* is a compiler directive that allows you to provide additional information to the compiler. This information can change compilation details that are not otherwise under your control. For example, the *pack* pragma affects the layout of data within a structure. Compiler pragmas are also called *directives*.

The preprocessor keyword *pragma* is part of the C++ standard, but the form, content, and meaning of pragmas is different for every compiler. No pragmas are defined by the C++ standard.

---

**Note** – Code that depends on pragmas is not portable.

## B.1 Pragma Forms

The various forms of a C++ compiler pragma are:

```plaintext
#pragma  keyword
#pragma  keyword ( a [ , a ] ...) [ , keyword ( a [ , a ] ...) ] , ...
#pragma sun keyword
```

The variable *keyword* identifies the specific directive; *a* indicates an argument.

### B.1.1 Overloaded Functions as Pragma Arguments

Several pragmas listed in this appendix take function names as arguments. In the event that the function is overloaded, the pragma uses the function declaration immediately preceding the pragma as its argument. Consider the following example:

```plaintext
int bar(int);
int foo(int);
```
int foo(double);
#pragma does_not_read_global_data(foo, bar)

In this example, foo means foo(double), the declaration of foo immediately preceding the pragma, and bar means bar(int), the only declared bar. Now, consider this following example in which foo is again overloaded:

int foo(int);
int foo(double);
int bar(int);
#pragma does_not_read_global_data(foo, bar)

In this example, bar means bar(int), the only declared bar. However, the pragma will not know which version of foo to use. To correct this problem, you must place the pragma immediately following the definition of foo that you want the pragma to use.

The following pragmas use the selection method described in this section:

- does_not_read_global_data
- does_not_return
- does_not_write_global_data
- no_side_effect
- opt
- rarely_called
- returns_new_memory

## B.2 Pragma Reference

This section describes the pragma keywords that are recognized by the C++ compiler.

### B.2.1 #pragma align

#pragma align integer(variable [,variable...])

Use align to make the listed variables memory-aligned to integer bytes, overriding the default. The following limitations apply:

- integer must be a power of 2 between 1 and 128; valid values are 1, 2, 4, 8, 16, 32, 64, and 128.
- variable is a global or static variable; it cannot be a local variable or a class member variable.
- If the specified alignment is smaller than the default, the default is used.
- The pragma line must appear before the declaration of the variables that it mentions; otherwise, it is ignored.
- Any variable mentioned on the pragma line but not declared in the code following the pragma line is ignored. Variables in the following example are properly declared.
#pragma align 64 (aninteger, astring, astruct)
int aninteger;
static char astring[256];
struct S {int a; char *b;} astruct;

When #pragma align is used inside a namespace, mangled names must be used. For example, in the following code, the #pragma align statement will have no effect. To correct the problem, replace a, b, and c in the #pragma align statement with their mangled names.

namespace foo {
    #pragma align 8 (a, b, c)
    static char a;
    static char b;
    static char c;
}

B.2.2 #pragma does_not_read_global_data

#pragma does_not_read_global_data(funcname [, funcname])

This pragma asserts that the specified routines do not read global data directly or indirectly. This allows for better optimization of code around calls to such routines. In particular, assignment statements or stores could be moved around such calls.

This pragma is permitted only after the prototype for the specified functions are declared. If the assertion about global access is not true, then the behavior of the program is undefined.

For a more detailed explanation of how the pragma treats overloaded function names as arguments, see "B.1.1 Overloaded Functions asPragma Arguments" on page 363.

B.2.3 #pragma does_not_return

#pragma does_not_return(funcname [, funcname])

This pragma is an assertion to the compiler that the calls to the specified routines will not return. This allows the compiler to perform optimizations consistent with that assumption. For example, register life-times terminate at the call sites which in turn allows more optimizations.

If the specified function does return, then the behavior of the program is undefined.

This pragma is permitted only after the prototype for the specified functions are declared as the following example shows:

extern void exit(int);
#pragma does_not_return(exit)
extern void __assert(int);
#pragma does_not_return(__assert)

For a more detailed explanation of how the pragma treats overloaded function names as arguments, see “B.1.1 Overloaded Functions asPragma Arguments” on page 363.

B.2.4  
#pragma does_not_write_global_data
#pragma does_not_write_global_data(funcname [, funcname])

This pragma asserts that the specified list of routines do not write global data directly or indirectly. This allows for better optimization of code around calls to such routines. In particular, assignment statements or stores could be moved around such calls.

This pragma is permitted only after the prototype for the specified functions are declared. If the assertion about global access is not true, then the behavior of the program is undefined.

For a more detailed explanation of how the pragma treats overloaded function names as arguments, see “B.1.1 Overloaded Functions asPragma Arguments” on page 363.

B.2.5  
#pragma dumpmacros
#pragma dumpmacros (value[, value...])

Use this pragma when you want to see how macros are behaving in your program. This pragma provides information such as macro defines, undefines, and instances of usage. It prints output to the standard error (stderr) based on the order macros are processed. The dumpmacros pragma is in effect through the end of the file or until it reaches a #pragma end_dumpmacro. See “B.2.6 #pragma end_dumpmacros” on page 367. You can substitute the following arguments in place of value:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>defs</td>
<td>Print all macro defines</td>
</tr>
<tr>
<td>undefs</td>
<td>Print all macro undefines</td>
</tr>
<tr>
<td>use</td>
<td>Print information about the macros used</td>
</tr>
<tr>
<td>loc</td>
<td>Print location (path name and line number) also for defs, undefs, and use</td>
</tr>
<tr>
<td>conds</td>
<td>Print use information for macros used in conditional directives</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>sys</td>
<td>Print all macros defines, undefines, and use information for macros in system header files</td>
</tr>
</tbody>
</table>

**Note** – The sub-options loc, conds, and sys are qualifiers fordefs, undefs and use options. By themselves, loc, conds, and sys have no effect. For example, `#pragma dumpmacros=loc,conds,sys` has no effect.

The `dumpmacros` pragma has the same effect as the command line option, however, the pragma overrides the command line option. See “A.2.123 - `xdumpmacros=[value, value...]`” on page 296.

The `dumpmacros` pragma does not nest so the following lines of code stop printing macro information when the `#pragma end_dumpmacros` is processed:

```c
#pragma dumpmacros (defs, undefs)
#pragma dumpmacros (defs, undefs)
...
#pragma end_dumpmacros
```

The effect of the `dumpmacros` pragma is cumulative. The following lines

```c
#pragma dumpmacros(defs, undefs)
#pragma dumpmacros(loc)
```

have the same effect as

```c
#pragma dumpmacros(defs, undefs, loc)
```

If you use the option `#pragma dumpmacros=use,no%loc`, the name of each macro that is used is printed only once. If you use the option `#pragma dumpmacros=use,loc` the location and macro name is printed every time a macro is used.

### B.2.6 `#pragma end_dumpmacros`

This pragma marks the end of a `dumpmacros` pragma and stops printing information about macros. If you do not use an end_`dumpmacros` pragma after a `dumpmacros` pragma, the `dumpmacros` pragma continues to generate output through the end of the file.
### B.2.7 #pragma fini

#pragma fini (identifier[, identifier...])

Use fini to mark identifier as a finalization function. Such functions are expected to be of type void, to accept no arguments, and to be called either when a program terminates under program control or when the containing shared object is removed from memory. As with initialization functions, finalization functions are executed in the order processed by the link editor.

In a source file, the functions specified in #pragma fini are executed after the static destructors in that file. You must declare the identifiers before using them in the pragma.

### B.2.8 #pragma hdrstop

Embed the hdrstop pragma in your source-file headers to identify the end of the viable source prefix. For example, consider the following files:

```plaintext
example% cat a.cc
#include "a.h"
#include "b.h"
#include "c.h"
#include <stdio.h>
#include "d.h"

example% cat b.cc
#include "a.h"
#include "b.h"
#include "c.h"
```

The viable source prefix ends at c.h so you would insert a #pragma hdrstop after c.h in each file.

#pragma hdrstop must only appear at the end of the viable prefix of a source file that is specified with the CC command. Do not specify #pragma hdrstop in any include file.

See "A.2.162 - xpch=v" on page 328 and "A.2.163 - xpchstop=file" on page 331.

### B.2.9 #pragma ident

#pragma ident string

Use ident to place string in the .comment section of the executable.
B.2.10  

#pragma init

#pragma init(identifier[, identifier...])

Use init to mark identifier as an initialization function. Such functions are expected to be of type void, to accept no arguments, and to be called while constructing the memory image of the program at the start of execution. Initializers in a shared object are executed during the operation that brings the shared object into memory, either at program start up or during some dynamic loading operation, such as dlopen(). The only ordering of calls to initialization functions is the order in which they are processed by the link editors, both static and dynamic.

Within a source file, the functions specified in #pragma init are executed after the static constructors in that file. You must declare the identifiers before using them in the pragma.

B.2.11  

#pragma must_have_frame

#pragma must_have_frame(funcname [,funcname])

This pragma requests that the specified list of functions always be compiled to have a complete stack frame (as defined in the System V ABI). You must declare the prototype for a function before listing that function with this pragma.

extern void foo(int);
extern void bar(int);
#pragma must_have_frame(foo, bar)

This pragma is permitted only after the prototype for the specified functions is declared. The pragma must precede the end of the function.

void foo(int) {
    .
    #pragma must_have_frame(foo)
    .
    return;
}

See “B.1.1 Overloaded Functions as Pragma Arguments” on page 363

B.2.12  

#pragma no_side_effect

#pragma no_side_effect(name[, name...])

Use no_side_effect to indicate that a function does not change any persistent state. The pragma declares that the named functions have no side effects of any kind. This means that the functions return result values that depend on the passed arguments only. In addition, the functions and their called descendants:
- Do not access for reading or writing any part of the program state visible in the caller at the point of the call.
- Do not perform I/O.
- Do not change any part of the program state not visible at the point of the call.

The compiler can use this information when doing optimizations.

If the function does have side effects, the results of executing a program which calls this function are undefined.

The name argument specifies the name of a function within the current translation unit. The pragma must be in the same scope as the function and must appear after the function declaration. The pragma must be before the function definition.

For a more detailed explanation of how the pragma treats overloaded function names as arguments, see “B.1.1 Overloaded Functions asPragma Arguments” on page 363.

B.2.13  \#pragma opt

\#pragma opt level (funcname[,funcname])

funcname specifies the name of a function defined within the current translation unit. The value of level specifies the optimization level for the named function. You can assign optimization levels 0, 1, 2, 3, 4, 5. You can turn off optimization by setting level to 0. The functions must be declared with a prototype or empty parameter list prior to the pragma. The pragma must proceed the definitions of the functions to be optimized.

The level of optimization for any function listed in the pragma is reduced to the value of -xmaxopt. The pragma is ignored when -xmaxopt=off.

For a more detailed explanation of how the pragma treats overloaded function names as arguments, see “B.1.1 Overloaded Functions asPragma Arguments” on page 363.

B.2.14  \#pragma pack(n)

\#pragma pack([n])

Use pack to affect the packing of structure members.

If present, n must be 0 or a power of 2. A value of other than 0 instructs the compiler to use the smaller of n-byte alignment and the platform’s natural alignment for the data type. For example, the following directive causes the members of all structures defined after the directive (and before subsequent pack directives) to be aligned no more strictly than on 2-byte boundaries, even if the normal alignment would be on 4- or 8-byte boundaries.
#pragma pack(2)

When \( n \) is 0 or omitted, the member alignment reverts to the natural alignment values.

If the value of \( n \) is the same as or greater than the strictest alignment on the platform, the directive has the effect of natural alignment. The following table shows the strictest alignment for each platform.

### TABLE B–1  Strictest Alignment by Platform

<table>
<thead>
<tr>
<th>Platform</th>
<th>Strictest Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86</td>
<td>4</td>
</tr>
<tr>
<td>SPARC generic, V8, V8a, V8plus, V8plusa, V8plusb</td>
<td>8</td>
</tr>
<tr>
<td>SPARC V9, V9a, V9b</td>
<td>16</td>
</tr>
</tbody>
</table>

A pack directive applies to all structure definitions which follow it, until the next pack directive. If the same structure is defined in different translation units with different packing, your program may fail in unpredictable ways. In particular, you should not use a pack directive prior to including a header defining the interface of a precompiled library. The recommended usage is to place the pack directive in your program code, immediately before the structure to be packed, and to place `#pragma pack(1)` immediately after the structure.

When using `#pragma pack` on a SPARC platform to pack denser than the type's default alignment, the `-malign` option must be specified for both the compilation and the linking of the application. The following table shows the storage sizes and default alignments of the integral data types.

### TABLE B–2  Storage Sizes and Default Alignments in Bytes

<table>
<thead>
<tr>
<th>Type</th>
<th>SPARCV8 Size, Alignment</th>
<th>SPARCV9 Size, Alignment</th>
<th>x86 Size, Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>1, 1</td>
<td>1, 1</td>
<td>1, 1</td>
</tr>
<tr>
<td>char</td>
<td>1, 1</td>
<td>1, 1</td>
<td>1, 1</td>
</tr>
<tr>
<td>short</td>
<td>2, 2</td>
<td>2, 2</td>
<td>2, 2</td>
</tr>
<tr>
<td>wchar_t</td>
<td>4, 4</td>
<td>4, 4</td>
<td>4, 4</td>
</tr>
<tr>
<td>int</td>
<td>4, 4</td>
<td>4, 4</td>
<td>4, 4</td>
</tr>
<tr>
<td>long</td>
<td>4, 4</td>
<td>8, 8</td>
<td>4, 4</td>
</tr>
<tr>
<td>float</td>
<td>4, 4</td>
<td>4, 4</td>
<td>4, 4</td>
</tr>
</tbody>
</table>
### TABLE B–2 Storage Sizes and Default Alignments in Bytes (Continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>SPARC V8 Size, Alignment</th>
<th>SPARC V9 Size, Alignment</th>
<th>x86 Size, Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>8, 8</td>
<td>8, 8</td>
<td>8, 4</td>
</tr>
<tr>
<td>long double</td>
<td>16, 8</td>
<td>16, 16</td>
<td>12, 4</td>
</tr>
<tr>
<td>pointer to data</td>
<td>4, 4</td>
<td>8, 8</td>
<td>4, 4</td>
</tr>
<tr>
<td>pointer to function</td>
<td>4, 4</td>
<td>8, 8</td>
<td>4, 4</td>
</tr>
<tr>
<td>pointer to member data</td>
<td>4, 4</td>
<td>8, 8</td>
<td>4, 4</td>
</tr>
<tr>
<td>pointer to member function</td>
<td>8, 4</td>
<td>16, 8</td>
<td>8, 4</td>
</tr>
</tbody>
</table>

#### B.2.15 #pragma rarely_called

```c
#pragma rarely_called(funcname[,funcname])
```

This pragma provides a hint to the compiler that the specified functions are called infrequently. This allows the compiler to perform profile-feedback style optimizations on the call-sites of such routines without the overhead of a profile-collections phase. Since this pragma is a suggestion, the compiler may not perform any optimizations based on this pragma.

The `#pragma rarely_called` preprocessor directive is only permitted after the prototype for the specified functions are declares. The following is an example of `#pragma rarely_called`:

```c
extern void error (char *message);
#pragma rarely_called(error)
```

For a more detailed explanation of how the pragma treats overloaded function names as arguments, see “B.1.1 Overloaded Functions asPragma Arguments” on page 363.

#### B.2.16 #pragma returns_new_memory

```c
#pragma returns_new_memory(name[,name...])
```

This pragma asserts that each named function returns the address of newly allocated memory and that the pointer does not alias with any other pointer. This information allows the optimizer to better track pointer values and to clarify memory location. This results in improved scheduling and pipelining.

If the assertion is false, the results of executing a program which calls this function are undefined.
The `name` argument specifies the name of a function within the current translation unit. The pragma must be in the same scope as the function and must appear after the function declaration. The pragma must be before the function definition.

For a more detailed explanation of how the pragma treats overloaded function names as arguments, see "B.1.1 Overloaded Functions asPragma Arguments" on page 363.

### B.2.17 #pragma unknown_control_flow

```
#pragma unknown_control_flow(name[, name...])
```

Use `unknown_control_flow` to specify a list of routines that violate the usual control flow properties of procedure calls. For example, the statement following a call to `setjmp()` can be reached from an arbitrary call to any other routine. The statement is reached by a call to `longjmp()`.

Because such routines render standard flowgraph analysis invalid, routines that call them cannot be safely optimized; hence, they are compiled with the optimizer disabled.

If the function name is overloaded, the most recently declared function is chosen.

### B.2.18 #pragma weak

```
#pragma weak name1 [= name2]
```

Use `weak` to define a weak global symbol. This pragma is used mainly in source files for building libraries. The linker does not warn you if it cannot resolve a weak symbol.

The `weak` pragma can specify symbols in one of two forms:

- **String form.** The string must be the mangled name for a C++ variable or function. The behavior for an invalid mangled name reference is unpredictable. The back end may or may not produce an error for invalid mangled name references. Regardless of whether it produces an error, the behavior of the back end when invalid mangled names are used is unpredictable.

- **Identifier form.** The identifier must be an unambiguous identifier for a C++ function that was previously declared in the compilation unit. The identifier form cannot be used for variables. The front end (`ccfe`) will produce an error message if it encounters an invalid identifier reference.

### B.2.18.1 #pragma weak name

In the form `#pragma weak name`, the directive makes `name` a weak symbol. The linker will not complain if it does not find a symbol definition for `name`. It also does not complain about multiple weak definitions of the symbol. The linker simply takes the first one it encounters.
If another compilation unit has a strong definition for the function or variable, name will be linked to that. If there is no strong definition for name, the linker symbol will have a value of 0.

The following directive defines ping to be a weak symbol. No error messages are generated if the linker cannot find a definition for a symbol named ping.

```cpp
#pragma weak ping
```

```cpp
#pragma weak name1 = name2
```

In the form `#pragma weak name1 = name2`, the symbol `name1` becomes a weak reference to `name2`. If `name1` is not defined elsewhere, `name1` will have the value `name2`. If `name1` is defined elsewhere, the linker uses that definition and ignores the weak reference to `name2`. The following directive instructs the linker to resolve any references to bar if it is defined anywhere in the program, and to foo otherwise.

```cpp
#pragma weak bar = foo
```

In the identifier form, `name2` must be declared and defined within the current compilation unit. For example:

```cpp
extern void bar(int) {...}
extern void _bar(int);
#pragma weak _bar=bar
```

When you use the string form, the symbol does not need to be previously declared. If both `bar` and `_bar` in the following example are `extern "C"`, the functions do not need to be declared. However, `bar` must be defined in the same object.

```cpp
extern "C" void bar(int) {...}
#pragma weak "_bar" = "bar"
```

### Overloading Functions

When you use the identifier form, there must be exactly one function with the specified name in scope at the pragma location. Attempting to use the identifier form of `#pragma weak` with an overloaded function is an error. For example:

```cpp
int bar(int); float bar(float);
#pragma weak bar // error, ambiguous function name
```

To avoid the error, use the string form, as shown in the following example.

```cpp
int bar(int); float bar(float);
#pragma weak "__1cDbar6Fi_i_" // make float bar(int) weak
```
See the Solaris Linker and Libraries Guide for more information.
<table>
<thead>
<tr>
<th><strong>Glossary</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ABI</strong></td>
</tr>
<tr>
<td><strong>abstract class</strong></td>
</tr>
<tr>
<td><strong>abstract method</strong></td>
</tr>
<tr>
<td><strong>ANSI C</strong></td>
</tr>
<tr>
<td><strong>ANSI/ISO C++</strong></td>
</tr>
<tr>
<td><strong>application binary interface</strong></td>
</tr>
<tr>
<td><strong>array</strong></td>
</tr>
<tr>
<td><strong>base class</strong></td>
</tr>
<tr>
<td><strong>binary compatibility</strong></td>
</tr>
<tr>
<td><strong>binding</strong></td>
</tr>
<tr>
<td><strong>cfront</strong></td>
</tr>
<tr>
<td><strong>class</strong></td>
</tr>
<tr>
<td><strong>class template</strong></td>
</tr>
<tr>
<td><strong>class variable</strong></td>
</tr>
<tr>
<td><strong>compiler option</strong></td>
</tr>
</tbody>
</table>
constructor

A special class member function that is automatically called by the compiler whenever a class object is created to ensure the initialization of that object’s instance variables. The constructor must always have the same name as the class to which it belongs. See destructor.

data member

An element of a class that is data, as opposed to a function or type definition.

data type

The mechanism that allows the representation of, for example, characters, integers, or floating-point numbers. The type determines the storage that is allocated to a variable and the operations that can be performed on the variable.

derived class

See inheritance.

destructor

A special class member function that is automatically called by the compiler whenever a class object is destroyed or the operator delete is applied to a class pointer. The destructor must always have the same name as the class to which it belongs, preceded by a tilde (~). See constructor.

dynamic binding

Connection of the function call to the function body at runtime. Occurs only with virtual functions. Also called late binding, runtime binding.

dynamic cast

A safe method of converting a pointer or reference from its declared type to any type that is consistent with the dynamic type to which it refers.

dynamic type

The actual type of an object that is accessed by a pointer or reference that might have a different declared type.

early binding

See static binding.

ELF file

Executable and Linking Format file, which is produced by the compiler.

exception

An error occurring in the normal flow of a program that prevents the program from continuing. Some reasons for errors include memory exhaustion or division by zero.

exception handler

Code specifically written to deal with errors, and that is invoked automatically when an exception occurs for which the handler has been registered.

exception handling

An error recovery process that is designed to intercept and prevent errors. During the execution of a program, if a synchronous error is detected, control of the program returns to an exception handler that was registered at an earlier point in the execution, and the code containing the error is bypassed.

flag

See compiler option.

function overloading

Giving the same name, but different argument types and numbers, to different functions. Also called functional polymorphism.

function prototype

A declaration that describes the function’s interface with the rest of the program.

function template

A mechanism that allows you to write a single function that you can then use as a model, or pattern, for writing related functions.

functional polymorphism

See function overloading.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>idempotent</td>
<td>The property of a header file that including it many times in one translation unit has the same effect as including it once.</td>
</tr>
<tr>
<td>incremental linker</td>
<td>A linker that creates a new executable file by linking only the changed .o files to the previous executable.</td>
</tr>
<tr>
<td>inheritance</td>
<td>A feature of object-oriented programming that allows the programmer to derive new classes (derived classes) from existing ones (base classes). There are three kinds of inheritance: public, protected, and private.</td>
</tr>
<tr>
<td>inline function</td>
<td>A function that replaces the function call with the actual function code.</td>
</tr>
<tr>
<td>instance variable</td>
<td>Any item of data that is associated with a particular object. Each instance of a class has its own copy of the instance variables defined in the class. Also called field. See also class variable.</td>
</tr>
<tr>
<td>instantiation</td>
<td>The process by which a C++ compiler creates a usable function or object (instance) from a template.</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization.</td>
</tr>
<tr>
<td>K&amp;R C</td>
<td>The de facto C programming language standard that was developed by Brian Kernighan and Dennis Ritchie before ANSI C.</td>
</tr>
<tr>
<td>keyword</td>
<td>A word that has unique meaning in a programming language, and that can be used only in a specialized context in that language.</td>
</tr>
<tr>
<td>late binding</td>
<td>See dynamic binding.</td>
</tr>
<tr>
<td>linker</td>
<td>The tool that connects object code and libraries to form a complete, executable program.</td>
</tr>
<tr>
<td>local variable</td>
<td>A data item known within a block, but inaccessible to code outside the block. For example, any variable defined within a method is a local variable and cannot be used outside the method.</td>
</tr>
<tr>
<td>locale</td>
<td>A set of conventions that are unique to a geographical area and/or language, such as date, time, and monetary format.</td>
</tr>
<tr>
<td>lvalue</td>
<td>An expression that designates a location in memory at which a variable's data value is stored. Also, the instance of a variable that appears to the left of the assignment operator.</td>
</tr>
<tr>
<td>mangle</td>
<td>See name mangling.</td>
</tr>
<tr>
<td>member function</td>
<td>An element of a class that is a function, as opposed to a data definition or type definition.</td>
</tr>
<tr>
<td>method</td>
<td>In some object-oriented languages, another name for a member function.</td>
</tr>
<tr>
<td>multiple inheritance</td>
<td>Inheritance of a derived class directly from more than one base class.</td>
</tr>
<tr>
<td>multithreading</td>
<td>The software technology that enables the development of parallel applications, whether on single- or multiple-processor systems.</td>
</tr>
<tr>
<td>name mangling</td>
<td>In C++, many functions can share the same name, so name alone is not sufficient to distinguish different functions. The compiler solves this problem by name mangling—creating a unique name for the function that consists of some combination of the function name and its parameters—to enable type-safe linkage. Also called name decoration.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>namespace</td>
<td>A mechanism that controls the scope of global names by allowing the global space to be divided into uniquely named scopes.</td>
</tr>
<tr>
<td>operator overloading</td>
<td>The ability to use the same operator notation to produce different outcomes. A special form of function overloading.</td>
</tr>
<tr>
<td>optimization</td>
<td>The process of improving the efficiency of the object code that is generated by the compiler.</td>
</tr>
<tr>
<td>option</td>
<td>See compiler option.</td>
</tr>
<tr>
<td>overloading</td>
<td>To give the same name to more than one function or operator.</td>
</tr>
<tr>
<td>polymorphism</td>
<td>The ability of a pointer or reference to refer to objects whose dynamic type is different from the declared pointer or reference type.</td>
</tr>
<tr>
<td>pragma</td>
<td>A compiler preprocessor directive, or special comment, that instructs the compiler to take a specific action.</td>
</tr>
<tr>
<td>runtime binding</td>
<td>See dynamic binding.</td>
</tr>
<tr>
<td>runtime type identification (RTTI)</td>
<td>A mechanism that provides a standard method for a program to determine an object type during runtime.</td>
</tr>
<tr>
<td>rvalue</td>
<td>The variable that is located to the right of an assignment operator. The rvalue can be read but not altered.</td>
</tr>
<tr>
<td>scope</td>
<td>The range over which an action or definition applies.</td>
</tr>
<tr>
<td>stab</td>
<td>A symbol table entry that is generated in the object code. The same format is used in both a.out files and ELF files to contain debugging information.</td>
</tr>
<tr>
<td>stack</td>
<td>A data storage method by which data can be added to or removed from only the top of the stack, using a last-in, first-out strategy.</td>
</tr>
<tr>
<td>static binding</td>
<td>Connection of a function call to a function body at compile time. Also called early binding.</td>
</tr>
<tr>
<td>subroutine</td>
<td>A function. In Fortran, a function that does not return a value.</td>
</tr>
<tr>
<td>switch</td>
<td>See compiler option.</td>
</tr>
<tr>
<td>symbol</td>
<td>A name or label that denotes some program entity.</td>
</tr>
<tr>
<td>symbol table</td>
<td>A list of all identifiers that are present when a program is compiled, their locations in the program, and their attributes. The compiler uses this table to interpret uses of identifiers.</td>
</tr>
<tr>
<td>template database</td>
<td>A directory containing all configuration files that are needed to handle and instantiate the templates that are required by a program.</td>
</tr>
<tr>
<td>template options file</td>
<td>A user-provided file containing options for the compilation of templates, as well as source location and other information. The template options file is deprecated and should not be used.</td>
</tr>
<tr>
<td>template specialization</td>
<td>A specialized instance of a class template member function that overrides the default instantiation when the default cannot handle a given type adequately.</td>
</tr>
<tr>
<td>term</td>
<td>definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>trapping</td>
<td>Interception of an action, such as program execution, in order to take other action. The interception causes the temporary suspension of microprocessor operations and transfers program control to another source.</td>
</tr>
<tr>
<td>type</td>
<td>A description of the ways in which a symbol can be used. The basic types are <code>integer</code> and <code>float</code>. All other types are constructed from these basic types by collecting them into arrays or structures, or by adding modifiers such as pointer-to or constant attributes.</td>
</tr>
<tr>
<td>variable</td>
<td>An item of data named by an identifier. Each variable has a type, such as <code>int</code> or <code>void</code>, and a scope. See also class variable, instance variable, local variable.</td>
</tr>
<tr>
<td>VTABLE</td>
<td>A table that is created by the compiler for each class that contains virtual functions.</td>
</tr>
</tbody>
</table>
Index

Numbers and Symbols
\>\> extraction operator
   complex, 198
   \iostream, 176
__\uname-s\uname-r\ predefine macro, 218
+d, compiler option, 216
+e(0|1), compiler option, 222
$ identifier, allowing as noninitial, 228
! NOT operator, \iostream, 175, 179
+p, compiler option, 261
#pragma align, 364
#pragma does_not_read_global_data, 365
#pragma does_not_return, 365
#pragma does_not_write_global_data, 366
#pragma dumpmacros, 366-367
#pragma end_dumpmacros, 367
#pragma fini, 368
#pragma ident, 368
#pragma init, 369
#pragma keywords, 364
#pragma must_have_frame, 369
#pragma no_side_effect, 369, 370
#pragma opt, 370
#pragma pack, 370
#pragma rarely_called, 372
#pragma returns_new_memory, 372
#pragma unknown_control_flow, 373
#pragma weak, 373
+w, compiler option, 91, 273
+w2, compiler option, 274
-386, compiler option, 212
-486, compiler option, 212

A
  .a, file name suffix, 36, 203
-a, compiler option, 213
absolute value, complex numbers, 193
accessible documentation, 23
aliases, simplifying commands with, 44
alignments
   default, 371-372
   strictest, 371
anachronisms, disallowing, 227
angle, complex numbers, 193
anonymous class instance, passing, 68-69
applications
   linking multithreaded, 117, 123
   MT-safe, 121
   using MT-safe \iostream objects, 133-135
applicator, parameterized manipulators, 186
arithmetic library, complex, 193-194
__ARRAYNEW, predefine macro, 218
assembler, compilation component, 40
assembly language templates, 360
assignment, \iostream, 183
ATS: Automatic Tuning System, 331

B
  -Bbinding, compiler option, 104, 213
binary input, reading, 178
binary optimization, 285
_BBOOL, predefine macro, 218
bool type and literals, allowing, 228
### Index

buffer
- defined, 191
- flushing output, 176
- _BUILTIN_va_arg_incr_, predefined macro, 218

### C
- `.cpp`, file name suffixes, 35
- C++ man pages, accessing, 141
- C++ standard library, 140
  - components, 153
  - man pages, 141, 155-168
  - replacing, 148-152
  - RogueWave version, 153
- `.c`, file name suffixes, 35
- C, file name suffixes, 35
- C API (application programming interface)
  - creating libraries, 207
  - removing dependency on C++ runtime libraries, 207
- c_exception, complex class, 197
- C standard library header files, replacing, 152
- `-c`, compiler option, 37, 214
- C99 support, 309
- cache
  - directory, template, 36
  - used by optimizer, 287
- cast
  - `const` and `volatile`, 105-106
  - dynamic, 107-110
    - casting down, 108-110
    - casting to void*, 108
    - casting up, 108
  - `reinterpret_cast`, 106-107
  - `static_cast`, 107
- `.cc`, file name suffixes, 35
- cc compiler options (Continued)
  - `-xhwcpul`, 50, 301
  - `-xipo`, 50
  - `-xipo_archive`, 307
  - `-xlinkopt`, 50
  - `-xmemalign`
    - grouped by functionality, 50
  - `-xopenmp`, 50
  - `-xpagesize`, 50
  - `-xpagesize_heap`, 51
  - `-xpagesize_stack`, 50
  - `-xpg`, 51
  - `-xprofile`, 51
  - `-xvector`, 51
- CC pragma directives, 364
- CCadmin command, 91
- CCFLAGS, environment variable, 44
- cerr standard stream, 130, 171
- `-cg`, compiler option, 215
- char* extractor, 177
- char, signedness of, 289
- characters, reading single, 177-178
- cin standard stream, 130, 171
- class declaration specifier, 64
- class instance, anonymous, 68-69
- class libraries, using, 143-146
- class templates, 76-78
  - See also templates declaration, 77
  - definition, 77
  - incomplete, 76
  - member, definition, 77-78
  - parameter, default, 81
  - static data members, 78
  - using, 78
- classes
  - passing directly, 114-115
  - passing indirectly, 114
- clog standard stream, 130, 171
- code generation
  - inliner and assembler, compilation component, 40
  - options, 48-49
- code optimization, by using `-fast`, 225
- code optimizer, compilation component, 40

---

384 Sun Studio 12 Update 1: C++ User's Guide
command line
  options, unrecognized, 39
  recognized file suffixes, 35
-compat
  compiler option, 215
  default linked libraries, affect on, 142
compatibility mode
  iostream, 171
  libC, 171, 173
  libcomplex, 193
See also -compat, 215
Tools.h++, 141
compat, libraries, available modes for, 139
-.compat
  linking C++ libraries, modes for, 146
compilation, memory requirements, 42-43
compiler commentary in object file, reading with
  er_src utility, 286
compiler
  component invocation order, 39
  diagnosing, 39
  versions, incompatibility, 36
compiling and linking, 37
complex_error, definition, 197
complex_error, message, 196
complex number data type, 193
complex
  compatibility mode, 193
complex, constructors, 194
complex
  efficiency, 200
  error handling, 197
  header file, 194
  input/output, 198
  library, 140, 145-146, 193-194
  library, linking, 194
  man pages, 200-201
  mathematical functions, 196
  mixed-mode, 199
  operators, 195
  standard mode and libCstd, 193
complex, trigonometric functions, 197
configuration macro, 140
conjugate of a number, 193
const_cast operator, 105-106
constant strings in read-only memory, 228
constructors
  complex class, 194-195
  iostream, 172
  static, 205
cool tools URL, 331
copying
  files, 188
  stream objects, 183
cout, standard stream, 130, 171
__cplusplus, predefined macro, 71, 215, 218
.cpp, file name suffixes, 35
.cxx, file name suffixes, 35

D
.d file extension, 315
-D, compiler option, 48, 217
-D_REENTRANT, 123
-d, compiler option, 219-220
-dalign, compiler option, 220
data type, complex number, 193
__DATE__, predefined macro, 218
-DDEBUG, 97
debugger data format, 294
debugging
  options, 50-51, 51-52
  preparing programs for, 38, 243
dec, iostream manipulator, 184
declaration specifiers
  __global, 64
  __hidden, 64
  __symbolic, 64
  __thread, 65
default libraries, static linking, 147
default operators, using, 112-113
definition included model, 73
definition separate model, 74
definitions, searching template, 98
delete array forms, recognizing, 230
dependency, on C++ runtime libraries, removing, 207
destructors, static, 205
dlclose(), function call, 205
Index

dlopen(), function call, 204
documentation, accessing, 22-23
documentation index, 22
double, complex value, 194
-dryrun, compiler option, 39, 220-221
dwarf debugger-data format, 294
dynamic (shared) libraries, 148, 205-206, 213, 244
dynamic_cast operator, 107-110
-errtags compiler option, 223
-errwarn compiler option, 224
exceptions
and multithreading, 118
building shared libraries that have, 104
disabling, 102
disallowing, 228
functions, in overriding, 66
longjmp and, 104
predefined, 102-103
setjmp and, 104
shared libraries, 206
signal handlers and, 104
standard class, 103
standard header, 102
trapping, 240
explicit instances, 93
explicit keyword, recognizing, 230
export keyword, recognizing, 228
extension features, 63-70
allowing nonstandard code, 228
defined, 30
external
instances, 92
linkage, 93
extraction
char*, 177
defined, 191
operators, 176
user-defined iostream, 176-177
whitespace, 178-179

E
-E compiler option, 221
EDOM, errno setting, 198
elfdump, 293
endl, iostream manipulator, 184
ends, iostream manipulator, 184
enum
forward declarations, 66-67
incomplete, using, 67
scope qualifier, using name as, 67
environment variables
CCFLAGS, 44
LD_LIBRARY_PATH, 204
SUN_PROFDATA, 340
SUN_PROFDATA_DIR, 340
SUNWS_CACHE_NAME, 96
er_src utility, 286
ERANGE, errno setting, 198
errno
definition, 197
interaction with -fast, 227
-erroff compiler option, 222
error function, 175
error handling
complex, 197
input, 179
error messages
compiler version incompatibility, 36
complex_error, 196
linker, 37, 39
error
bits, 175
checking, MT-safety, 124
state, iostreams, 175

F
-fast, compiler option, 225-227
-features, compiler option, 107
-features, compiler option, 63, 102, 118, 227
_FILE_, predefined macro, 218
file descriptors, using, 182
file names
suffixes, 35
.SUNW.C file name suffix, 150
template definition files, 98
files
   See also source files
   C standard header files, 150
   copying, 181, 188
   executable program, 37
   multiple source, using, 36
   object, 37, 48, 205
   opening and closing, 181-182
   repositioning, 182-183
   standard library, 150
   using fstreams with, 180-183
   -filt, compiler option, 231
   finalization functions, 368
   -flags, compiler option, 233
   float inserter, istream output, 173
   floating point
      invalid, 240
      options, 52
      precision (Intel), 239
   flush, istream manipulator, 184
   flush, istream manipulator, 176
   -fnonstd, compiler option, 234
   -fns, compiler option, 234-236
   format control, istreams, 183
   Fortran runtime libraries, linking, 309
   -fprecision=p, compiler option, 236
   front end, compilation component, 40
   -fround=r, compiler option, 237-238
   -fdecimal=n, compiler option, 238
   -fstore, compiler option, 239
   fstream, defined, 172, 191
   fstream.h
      istream header file, 173
      using, 180
   -ftrap, compiler option, 240
   __func__, identifier, 69-70
   function-level reordering, 299
   function templates, 75-76
   See also templates
   declaration, 75
   definition, 75-76
   using, 76
   function, declaration specifier, 63
   functions, name in __func__, 69-70
   functions
      in dynamic (shared) libraries, 205
      inlining by optimizer, 302
      MT-safe public, 122
      overriding, 66
      static, as class friend, 69
      streambuf public virtual, 132

G
   -G
      dynamic library command, 205
      option description, 241
   -g
      compiling templates using, 97
      option description, 242
      garbage collection
         libraries, 141, 146
      get, char extractor, 177
      get pointer, streambuf, 188
   __global, 64
   global
      data, in a multithreaded application, 129
      instances, 93
      linkage, 93
      shared objects in MT application, 130
   -gO option description, 243

H
   -H, compiler option, 244
   -h, compiler option, 244
   header files
      C standard, 150
      complex, 200
      creating, 71-72
      idempotency, 72
      istream, 130, 173, 184
      language-adaptable, 71-72
      standard library, 149, 154-155
   heap, setting page size for, 325
   -help, compiler option, 244
   hex, istream manipulator, 184
Index

__hidden, 64

I
.i, file name suffixes, 35
I/O library, 171
-I-, compiler option, 245-247
-I, compiler option, 98, 244-245
-i, compiler option, 247
__i386, predefined macro, 219
i386, predefined macro, 219
idempotency, 71
ifstream, defined, 172
.il, file name suffixes, 36
.include, compiler option, 248
include directories, template definition files, 98
include files, search order, 244, 245-247
incompatibility, compiler versions, 36
initialization function, 369
-inline, See -xinline, 248
inline expansion, assembly language templates, 40
inline functions
  by optimizer, 302
  C++, when to use, 112
input/output, complex, 171, 198
input
  binary, 178
  error handling, 179
  iostream, 176
  peeking at, 178
insertion operator
  complex, 198
  iostream, 173-176
insertion
  defined, 191
  operator, 173
instance methods
  explicit, 95
  global, 95
  semi-explicit, 96
  static, 95
  template, 92
instance states, consistent, 97
-instances=a, compiler option, 92, 248
instantiation
  options, 92
  template class static data members, 80
  template classes, 79
  template function members, 80
  template functions, 79
-instlib, compiler option, 249
intermediate language translator, compilation component, 40
internationalization, implementation, 31
interprocedural optimizations, 305
interval arithmetic libraries, linking, 302
iomanip.h, iostream header files, 173, 184
iostream.h, iostream header file, 130, 173
iostream
  classic iostreams, 140, 143, 255
  compatibility mode, 171
  constructors, 172
  copying, 183
  creating, 180-183
  defined, 191
  error bits, 175
  error handling, 179
  extending functionality, MT considerations, 132
  flushing, 176
  formats, 183
  header files, 173
  input, 176
  library, 140, 143-144, 146
  man pages, 171, 189-190
  manipulators, 183-187
  mixing old and new forms, 255
  MT-safe interface changes, 127-129
  iostream, MT-safe reentrant functions, 123
iostream
  MT-safe restrictions, 124-125
  new class hierarchy for MT, 127-128
  new MT interface functions, 128
  output errors, 175
  output to, 173-176
  predefined, 171
  single-threaded applications, 124
  standard iostreams, 140, 143, 255
  standard mode, 171, 173, 255

388  Sun Studio 12 Update 1: C++ User’s Guide
iostream (Continued)
  stdio, 179-180, 187
  stream assignment, 183
  structure, 172
  terminology, 190-191
  using, 173-180
ISO C++ standard
  conformance, 30
  one-definition rule, 87, 97
ISO10646 UTF-16 string literal, 358
istream class, defined, 172
istream class, defined, 172

K
  -Kkeeptmp, compiler option, 251
  -Kpic, compiler option, 205, 251
  -KPic, compiler option, 205, 250

L
  -L, compiler option, 142, 251
  -l, compiler option, 48, 139, 142, 251
languages
  C99 support, 309
  options, 52-53
  support for native, 31
LD_LIBRARY_PATH environment variable, 204
left-shift operator
  complex, 198
  iostream, 173
libc library, 139
libc
  compatibility mode, 171, 173
  compiling and linking MT-safety, 123-124
  library, 140
  MT environment, using in, 121
  new MT classes, 127
libCrun library, 117, 118, 140, 142
libCstd library, See C++ standard library
libCsunimath, library, 140
libdemangle library, 140
libgc library, 140
libiostream, See iostream
  -libmeee, compiler option, 252
  -libmil, compiler option, 252
libm
  inline templates, 312
  library, 139
  optimized version, 312
libraries, building
  dynamic (shared), 203
  for private use, 206
  for public use, 206
  linking options, 242
  shared with exceptions, 206
  static (archive), 203-208
  with C API, 207
libraries, building
  shared libraries, 293
  C++ compiler, provided with, 139
  C++ standard, 153
  C interface, 139
  class, using, 143
  classic iostream, 171
  configuration macro, 140
  interval arithmetic, 302
  linking options, 53-54, 146
  linking order, 48
  linking with -mt, 139
  naming a shared library, 244
  optimized math, 312
  replacing, C++ standard library, 148-152
  shared, 148, 219
  suffixes, 203
  Sun Performance Library, linking, 253, 312
  understanding, 203-204
  using, 139-152
library, compiler option, 142, 146, 147, 252-256
libSwtool, See Tools.h++, 141
libthread library, 139
limit, command, 43
__LINE__, predefined macro, 218
link-time optimization, 313
link-time options, list of, 50
linker scoping, 63
Index

linking
  complex library, 145-146
  consistent with compilation, 38
  disabling system libraries, 319
  dynamic (shared) libraries, 204, 213
  iostream library, 144
  libraries, 139, 142, 146
  library options, 53-54
    -mt option, 123
  MT-safe libC library, 123-124
  separate from compilation, 37
  static (archive) libraries, 142, 147, 203, 213, 266
  symbolic, 150
  template instance methods, 92
  literal strings in read-only memory, 228
  local-scope rules, enabling and disabling, 228
locking
  See also stream locker
    mutex, 126, 132
    object, 130-132
    streambuf, 122
loops, 295
  reduction with --xreduction, 343
  --xloopinfo, 314
-lthread
  suppressed by -xnolib, 147
  using -mt in place of, 117, 123

M
macros
  predefined, 218-219
  See also individual macros under alphabetical listings, 218-219
magnitude, complex numbers, 193
makefile dependencies, 315
man pages
  accessing, 31, 141
  C++ standard library, 155-168
    complex, 200-201
    iostream, 171, 181, 183, 186
manipulators
  iostreams, 183-187
  plain, 185

manipulators (Continued)
  predefined, 184
  math.h, complex header files, 200
  math library, optimized version, 312
  mathematical functions, complex arithmetic
    library, 196
  -mc, compiler option, 257
  member variables, caching, 115
  memory requirements, 42-43
  -migration, compiler option, 257
  -misalign, compiler option, 257-258
  mixed-language linking, 309
  mixed-mode, complex arithmetic library, 199
  -mr, compiler option, 258
  -mt compiler option
    and libthread, 123
    linking libraries, 139
    option description, 258
MT-safe
  applications, 121
  classes, considerations for deriving, 132
  library, 121
  object, 121
  performance overhead, 126, 127
  public functions, 122
  multimedia types, handling of, 360
  multiple source files, using, 36
  multithreaded
    application, 117
    compilation, 117-118
    exception-handling, 118
mutable keyword, recognizing, 228
mutex locks, MT-safe classes, 126, 132
mutual exclusion region, defining a, 132

N
namespace keyword, recognizing, 230
  -native, compiler option, 259
native-language support, application development, 31
new array forms, recognizing, 230
  -noex, compiler option, 118, 259
  -nofstore, compiler option, 259
  -nolib, compiler option, 143, 259
-nolibmil, compiler option, 259
nonincremental link editor, compilation
  component, 40
nonstandard features, 63-70
  allowing nonstandard code, 228
defined, 30
-noqueue, compiler option, 259
-norunpath, compiler option, 143, 259-260
nsplings, alternative, 227
numbers, complex, 193

O
.o files
  option suffixes, 36
  preserving, 37
-0, compiler option, 260
-0level, compiler option, 260
-o, compiler option, 260
object files
  linking order, 48
  reading compiler commentary with er_src, 286
  relocatable, 205
object thread, private, 131
objects
  destruction of shared, 133
  destruction order, 229
  global shared, 130
  strategies for dealing with shared, 130
  stream_locker, 132
  temporary, 111
  temporary, lifetime of, 229
  within library, when linked, 203
oct, iostream manipulator, 184
ofstream class, 180
operators
  basic arithmetic, 195
  complex, 198
  iostream, 173-176, 176-177
  scope resolution, 126
optimization
  at link time, 313
  levels, 321
  math library, 312
  (Continued)
optimization
  options for, 56-58
  target hardware, 348
  with -fast, 225
  with -xmaxopt, 316
  with pragma opt, 370
  optimizer out of memory, 43
options
  code generation, 48-49
  debugging, 50-51, 51-52
  description subsections, 211-212
  expansion compilation, 225-226
  floating point, 52
  language, 52-53
  library, 142-143
  library linking, 53-54
  obsolete, 54-55, 263
  optimization, 56-58
  output, 55-56, 56
  performance, 56-58
  preprocessor, 58
  processing order, 35, 48
  profiling, 58-59
  reference, 59
See also individual options under alphabetical
  listings, 47-60
source, 59
overflow
  function, streambuf, 132
overhead, MT-safe class performance, 126, 127
Index

P
-P, compiler option, 261
page size, setting for stack or heap, 325
parallelization, with --xreduction, 343
parallelization
  turning on warning messages, 361
  turning on with -xautopar for multiple
  processors, 285
parameterized manipulators, iostreams, 186-187
PEC: Portable Executable Code, 331
peeking at input, 178
Pentium, 354
-pentium, compiler option, 261
performance
  optimizing with -fast, 225
  options, 56-58
  overhead of MT-safe classes, 126, 127
-pg, compiler option, 262
-PIC, compiler option, 262
-pic, compiler option, 262
placement, template instances, 92
plain manipulators, iostreams, 185
polar, complex number, 193
precedence, avoiding problems of, 174
precompiled-header file, 328
predefined macros, 218-219
predefined manipulators, iomanip.h, 184
prefetch instructions, enabling, 335
preprocessor
  defining macro to, 217
  options, 58
preserving signedness of chars, 289
private, object thread, 131
processing order, options, 35
processor, specifying target, 348
profiling options, 58-59, 339
Programming Language–C++, standards conformance, 30
programs
  basic building steps, 33-34
  building multithreaded, 117-118
-ppta, compiler option, 262
ptclean command, 91
pthread_cancel() function, 118
-post, compiler option, 98, 262
-post, compiler option, 262
-pptr, compiler option, 263
-ptrv, compiler option, 263
public functions, MT-safe, 122
put pointer, streambuf, 188

Q
-qoption, compiler option, 263
-qoption, compiler option, 264
-qp, compiler option, 264
-qproduce, compiler option, 265
-qproduce, compiler option, 265

R
-R, compiler option, 143, 265
-readme, compiler option, 266
readme file, 30-31
real numbers, complex, 193
reference options, 59
reinterpret_cast operator, 106-107, 277
reorder functions, 299
repositioning within a file, fstream, 182-183
resetiosflags, iostream manipulator, 184
restricted pointers, 346
restrictions, MT-safe iostream, 124-125
right-shift operator
  complex, 198
  iostream, 176
RogueWave
  C++ standard library, 153
  See also Tools.h++, 141
rtti keyword, recognizing, 230
runtime error messages, 102
runtime libraries readme, 169

S
.s, file name suffixes, 35
.S, file name suffixes, 35
s new in this release, 27
- S, compiler option, 266
- s, compiler option, 266
- sb, compiler option, 266
- sbfast, compiler option, 266
sbufpub, man pages, 181
scope resolution operator, unsafe_classes, 126
search path

definitions, 98
dynamic library, 143
include files, defined, 244
source options, 59
standard header implementation, 150
template options, 59
searching, template definition files, 98
semi-explicit instances, 93, 96
sequences, MT-safe execution of I/O operations, 130
set_terminate() function, 118
set_unexpected() function, 118
setbase, iostream manipulator, 184
setfill, iostream manipulator, 184
setioflags, iostream manipulator, 184
setprecision, iostream manipulator, 184
setw, iostream manipulator, 184
shared libraries

accessing from a C program, 207-208
building, 205-206, 241
building, with exceptions, 104
containing exceptions, 206
disallowing linking of, 219
naming, 244
shared objects, 130, 133
shell, limiting virtual memory in, 43
shift operators, iostreams, 185
signal handlers

and exceptions, 101
and multithreading, 118
signedness of chars, 289
sizes, storage, 371-372
skip flag, iostream, 178
.so, file name suffix, 36, 203
.so., file name suffix, 36
Solaris operating environment libraries, 139
source compiler options, 59
source files

linking order, 48
location conventions, 98
__sparc, predefined macro, 218
sparc, predefined macro, 219
__sparcv9, predefined macro, 219
stabs debugger-data format, 294
stack, setting page size for, 325
Standard C++ Class Library Reference, 154
standard error, iostreams, 171
standard headers

implementing, 150-152
replacing, 151
standard input, iostreams, 171
standard iostream classes, 171
standard mode

iostream, 171, 173
libCstd, 193
See also -compat, 215
Tools.h++, 141
standard output, iostreams, 171
standard streams, iostream.h, 130
Standard Template Library (STL), 153
standards, conformance, 30
static (archive) libraries, 203
static_cast operator, 107
static data, in a multithreaded application, 129
static instances, 93
static linking

compiler provided libraries, 142, 266
default libraries, 147
library binding, 213
template instances, 94
static

functions, referencing, 87
- staticlib, compiler option, 142, 147, 266
static

objects, initializers for nonlocal, 228
variables, referencing, 87
__STDC__, predefined macro, 71, 218
stdio

stdiobuf man pages, 187
stdiostream.h, iostream header file, 173

393
Index

stdio
  with iostreams, 179-180
STL (Standard Template Library), components, 153
STLport, 168
storage sizes, 371-372
stream, defined, 191
stream.h, iostream header file, 173
stream_locker
  man pages, 132
  synchronization with MT-safe objects, 127
streambuf
  defined, 187, 191
  get pointer, 188
  locking, 122
  man pages, 188
  new functions, 128
  public virtual functions, 132
  put pointer, 188
  queue-like versus file-like, 188
  using, 188
streampos, 182
string literal of U"..." form, 358
strstream, defined, 172, 191
strstream.h, iostream header file, 173
struct, anonymous declarations, 67-68
structure declaration specifier, 64
subprograms, compilation options, 38
suffixes
  command line file name, 35
  files without, 150
  library, 203
  .SUNWCCch, 150
__sun, predefined macro, 219
__SUNPRO_CC, 218
__SUNPRO_CC, predefined macro, 218
__SUNPRO__CC_COMPAT=(4|5), predefined macro, 215, 218
.SUNWCCch file name suffix, 150
SunWS_cache, 96
__SVR4, predefined macro, 218
swap -s, command, 42
swap space, 42
symbol declaration specifier, 63
symbol tables, executable file, 266
__symbolic, 64
symbols, See macros
-sync_stdio, compiler option, 268
syntax
  CC commands, 34
  options, 47

T
-temp=dir, compiler option, 269
-template, compiler option, 98
-template, compiler option, 92, 269
template definition
  included, 73
  search path, 98
  separate, 74
  separate, file, 98
  troubleshooting a search for definitions, 98
template instantiation, 79-80
  explicit, 79-80
  function, 79-80
  implicit, 79
  whole-class, 92
template pre-linker, compilation component, 40
template problems, 83-89
  friend declarations of template functions, 84-86
  local types as arguments, 84
  non-local name resolution and instantiation, 83
  static objects, referencing, 87
  troubleshooting a search for definitions, 98
  using qualified names in template definitions, 86-87
templates
  cache directory, 36
  commands, 91
  compilation, 93
  definitions-separate vs. definitions-included organization, 98
  inline, 311
  instance methods, 92, 97
  linking, 38
  nested, 80
  options, 59-60
  partial specialization, 82
  repositories, 96
templates (Continued)
source files, 98
specialization, 81-82
Standard Template Library (STL), 153
static objects, referencing, 87
troubleshooting a search for definitions, 98
verbose compilation, 91
terminate() function, 118
thr_keycreate, man pages, 130
__thread, 65
thread local storage of variables, 65
thread options, 60
__TIME__, predefined macro, 218
--time, compiler option, 271
Tools.h++
classic and standard iostreams, 141
compiler options, 146
debug library, 140
documentation, 141
standard and compatibility mode, 141
trapping mode, 240
trigonometric functions, complex arithmetic library, 197
trigraph sequences, recognizing, 357

U
-U, compiler option, 48, 271
ulimit, command, 43
unexpected() function, 118
union declaration specifier, 64
__unix, predefined macro, 218
unix, predefined macro, 218
-unroll=n, compiler option, 271
user-defined types
iostream, 174
MT-safe, 125

V
-V, compiler option, 272
-v, compiler option, 39, 272
__VA_ARGS__ identifier, 41
value classes, using, 113-115
values
doUBLE, 194
float, 173
flush, 176
inserting on cout, 174
long, 187
manipulator, 173, 186
variable, thread-local storage specifier, 65
variable argument lists, 41
variable declaration specifier, 63
-vdelx, compiler option, 272
-verbose, compiler option, 91, 272
viable prefix, 329
virtual memory, limits, 42-43
VIS Software Developers Kit, 360

W
-w, compiler option, 274
warnings
anachronisms, 274
C header replacement, 152
inefficient code, 273
nonportable code, 273
problematic ARM language constructs, 229
suppressing, 274
technical violations reducing portability, 274
unrecognized arguments, 39
What', 27
whitespace
extractors, 178-179
leading, 178
skipping, 178, 185
workstations, memory requirements, 43
ws, iostream manipulator, 179
ws, iostream manipulator, 184

X
x inserter, iostream, 173
-xa, compiler option, 274-275
-xalias_level, compiler option, 275
-xannotate, compiler option, 278
-xar, compiler option, 93, 204, 278
-xarch=isa, compiler option, 279
-xautopar, compiler option, 285
-xbinopt, 285
-xbinopt, compiler option, 285
-xbuiltin, compiler option, 286
-xcache=c, compiler option, 287-288
-xcg, compiler option, 288
-xcg89, compiler option, 288
-xchar, compiler option, 289
-xcheck, compiler option, 290
-xchip=c, compiler option, 290
-xcode=a, compiler option, 292-294
-xdebugformat, 294
-xdepend, compiler option, 295
-xdumpmacros, compiler option, 296
-xe, compiler option, 299
-xF, compiler option, 299-300
-xhelp=flags, compiler option, 301
-xhelp=readme, compiler option, 301
-xhreadvar, compiler option, 356
-xhwprof, 301
-xia, compiler option, 302
-xinline, compiler option, 302
-xipo, compiler option, 305
-xipo_archive, 307
-xjobs, compiler option, 308
-xlang, compiler option, 309
-xldscope, compiler option, 63, 310
-xlibmieee, compiler option, 311
-xlibmili, compiler option, 311-312
-xlibmopt, compiler option, 312
-xlic_lib, compiler option, 312
-xlicinfo, compiler option, 312
-xlinkopt, compiler option, 313
-xloopinfo, compiler option, 314
-xM, compiler option, 314-315
-xM, compiler option, 274
-xM1, compiler option, 315
-xmaxopt, 316
-xmaxopt, compiler option, 316
-xMD, compiler option, 315
-xmemalign, compiler option, 316
-xMerge, compiler option, 316
-xMF, compiler option, 315
-xMD, compiler option, 316
-xmodel, compiler option, 318
-xnolib, compiler option, 143, 147, 319
-xnolibmil, compiler option, 320
-xnolibmopt, compiler option, 320-321
-xOlevel, compiler option, 321-324
-xopenmp, compiler option, 324
-xpagesize, compiler option, 325
-xpagesize_heap, compiler option, 326
-xpagesize_stack, compiler option, 327
-xspec, compiler option, 331
-xpg, compiler option, 331-332
-xport64, compiler option, 332
-xprefetch, compiler option, 335
-xprefetch_auto_type, compiler option, 338
-xprefetch_level, compiler option, 338
-xprofile, compiler option, 339-342
-xprofile ircache, compiler option, 342
-xprofile_pathmap, compiler option, 342
-xreduction, compiler option, 343
-xregs, compiler option, 206, 343-346
-xrestrict, compiler option, 346
-xs, compiler option, 347
-xsafe=mem, compiler option, 348
-xspace, compiler option, 348
-xtarget=t, compiler option, 348-355
-xt ime, compiler option, 357
-xtrigraphs, compiler option, 357
-xunroll=n, compiler option, 358
-xustr, compiler option, 358
-xvector, compiler option, 359
-xvis, compiler option, 360
-xvpara, compiler option, 361
-xwe, compiler option, 361

Z
-z arg, compiler option, 362