



C++ User's Guide

A Sun Microsystems, Inc.
Business
901 San Antonio Road
Palo Alto, , CA 94303-4900

Part No: 805-4954
Revision A, February 1999

USA 650 960-1300 fax 650 969-9131



C++ User's Guide

Part No: 805-4954
Revision A, February 1999

Copyright Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

All rights reserved. This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Portions of this product may be derived from the UNIX[®] system, licensed from Novell, Inc., and from the Berkeley 4.3 BSD system, licensed from the University of California. UNIX is a registered trademark in the United States and in other countries and is exclusively licensed by X/Open Company Ltd. Third-party software, including font technology in this product, is protected by copyright and licensed from Sun's suppliers. RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

Sun, Sun Microsystems, the Sun logo, SunDocs, SunExpress, Solaris, Sun Performance Library, Sun Performance WorkShop, Sun Performance WorkShop Fortran, Sun Visual WorkShop, Sun WorkShop, Sun WorkShop Compilers C, Sun WorkShop Compilers C++, Sun WorkShop Compilers Fortran, Sun WorkShop Memory Monitor, Sun WorkShop Professional, Sun WorkShop Professional C, and Sun WorkShop TeamWare are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK[®] and Sun[™] Graphical User Interfaces were developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox Corporation in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a nonexclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

Copyright 1997 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, Californie 94303-4900 U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie et la décompilation. Aucune partie de ce produit ou de sa documentation associée ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Des parties de ce produit pourront être dérivées du système UNIX[®] licencié par Novell, Inc. et du système Berkeley 4.3 BSD licencié par l'Université de Californie. UNIX est une marque enregistrée aux Etats-Unis et dans d'autres pays, et licenciée exclusivement par X/Open Company Ltd. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, et Solaris sont des marques déposées ou enregistrées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC, utilisées sous licence, sont des marques déposées ou enregistrées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

Les utilisateurs d'interfaces graphiques OPEN LOOK[®] et Sun[™] ont été développés de Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox Corporation pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique, cette licence couvrant aussi les licenciés de Sun qui mettent en place les utilisateurs d'interfaces graphiques OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" SANS GARANTIE D'AUCUNE SORTE, NI EXPRESSE NI IMPLICITE, Y COMPRIS, ET SANS QUE CETTE LISTE NE SOIT LIMITATIVE, DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DES PRODUITS A REpondre A UNE UTILISATION PARTICULIERE OU LE FAIT QU'ILS NE SOIENT PAS CONTREFAISANTS DE PRODUITS DE TIERS.



Contents

	Preface	xiv
1.	The C++ Compiler	1
	Standards Conformance	1
	Operating Environments	2
	READMEs	2
	Man Pages	3
	Licensing	3
	New Features of the C++ Compiler	3
	C++ Utilities	4
	Native-Language Support	5
2.	Using the C++ Compiler	7
	Getting Started	7
	Invoking the Compiler	8
	Command Syntax	9
	File Name Conventions	9
	Using Multiple Source Files	10
	Compiling and Linking	11
	Compile-Link Sequence	11
	Separate Compiling and Linking	11

Consistent Compiling and Linking	12
Compiling for SPARC V9	13
Understanding the Compiler Organization	14
Memory Requirements	16
Simplifying Commands	19
Using Aliases Within the C Shell	19
Using <code>CCFLAGS</code> to Specify Compile Options	19
Using <code>make</code>	20
3. C++ Compiler Options	23
Options Summarized by Function	24
Code Generation Options	24
Debugging Options	25
Floating-Point Options	27
Language Options	28
Library Options	28
Licensing Options	30
Obsolete Options	31
Output Options	31
Performance Options	32
Preprocessor Options	34
Profiling Options	35
Reference Options	35
Source Options	36
Template Options	36
Thread Options	37
How Option Information Is Organized	37
Option Reference	39
-386	39

-486 39
-a 39
-*Bbinding* 39
-c 40
-cg[89|92] 41
-compat[=(4|5)] 41
+d 42
-Dname[=*def*] 42
-d(*y*|*n*) 44
-dalign 44
-dryrun 44
-E 45
+e(0|1) 45
-fast 45
-features=*a*[...*a*] 47
-flags 49
-fnonstd 49
-fns[=(yes|no)] 50
-fprecision=*p* 51
-fround=*r* 52
-fsimple[=*n*] 53
-fstore 55
-ftrap=*t* 55
-G 57
-g 58
-g0 58
-H 59
-help 59

-hname 59
-i 59
-I*pathname* 60
-inline=*rlst* 60
-instances=*a* 60
-keptmp 61
-KPIC 62
-Kpic 62
-Ldir 62
-llib 62
-libmieee 63
-libmil 63
-library=*I[,...I]* 63
-migration 65
-misalign 66
-mt 66
-native 67
-noex 67
-nofstore 67
-nolib 67
-nolibmil 67
-noqueue 67
-norunpath 68
-O 68
-O*level* 68
-o *filename* 68
+p 69
-P 69

-p 69
-pentium 69
-pg 69
-PIC 70
-pic 70
-pta 70
-ptipath 70
-pto 70
-ptr*database-path* 70
-ptv 71
-Qoption *phase option*[...*option*] 71
-qoption *phase option* 72
-qp 72
-Qproduce *sourcetype* 72
-qproduce *sourcetype* 72
-R*pathname* 72
-readme 73
-S 74
-s 74
-sb 74
-sbfast 74
-staticlib=*I*[,...*I*] 74
-temp=*dir* 75
-template=*w* 76
-time 76
-U*name* 76
-unroll=*n* 77
-v 77

-v 77
-vdelx 77
-verbose=v[,...v] 77
+w 78
+w2 79
-w 79
-xa 79
-xar 80
-xarch=a 80
-xcache=c 84
-xcg(89|92) 85
-xchip=c 86
-xcode=a 87
-xF 88
-xhelp=flags 89
-xhelp=readme 89
-xildoff 89
-xildon 90
-xinline=f[,...f] 90
-xlibmieee 91
-xlibmil 91
-xlibmopt 91
-xlic_lib=l[,...l] 92
-xlicinfo 92
-Xm 93
-xM 93
-xM1 94
-xMerge 94

-xnolib 94
-xnolibmil 95
-xnolibmopt 96
-xO[*level*] 96
-xpg 99
-xprefetch[=(yes|no)] 99
-xprofile=*p* 99
-xregs=*r*[,...*r*] 102
-xs 103
-xsafe=mem 104
-xsb 104
-xsbfast 105
-xspace 105
-xtarget=*t* 105
-xtime 113
-xunroll=*n* 113
-xwe 113
-ztext 113

4. Compiling Templates 115

Verbose Compilation 115

Template Commands 115

Template Instance Placement and Linkage 116

External Instances 116

Static Instances 117

Global Instances 117

Explicit Instances 117

Semi-Explicit Instances 118

The Template Repository 119

	Repository Structure	119
	Writing to the Template Repository	119
	Reading From Multiple Template Repositories	119
	Sharing Template Repositories	120
	Template Definition Searching	120
	Source File Location Conventions	120
	Definitions Search Path	120
	Template Instance Automatic Consistency	121
	Compile-Time Instantiation	121
5.	Using Libraries	123
	The C Libraries	123
	Libraries Provided With the C++ Compiler	124
	C++ Library Descriptions	125
	Default C++ Libraries	125
	Related Library Options	126
	Using Class Libraries	127
	The <code>iostream</code> Library	127
	The <code>complex</code> Library	128
	Linking C++ Libraries	129
	Statically Linking Standard Libraries	130
	Using Shared Libraries	131
	Standard Header Implementation	132
6.	Building Libraries	135
	Understanding Libraries	135
	Building Static (Archive) Libraries	136
	Building Dynamic (Shared) Libraries	137
	Building Shared Libraries With Exceptions	138
	Building Libraries for Private Use	138

Building Libraries for Public Use	138
Building a Library With a C API	139
Using <code>dlopen</code> to Access a C++ Library From a C Program	139
Building Multithreaded Programs	140
Indicating Multithreaded Compilation	140
Using <code>libc</code> With Threads and Signals	140
Glossary	143
Index	149

Preface

This manual instructs you in the use of the C++ 5.0 compiler, and provides detailed information on command-line compiler options.

Who Should Use This Book

This manual is intended for programmers with a working knowledge of C++ and some understanding of the Solaris[™] operating environment and UNIX[®] commands.

How This Book Is Organized

This book contains the following chapters:

Chapter 1, "The C++ Compiler," gives an overview of the C++ compiler.

Chapter 2, "Using the C++ Compiler," provides instructions for invoking the compiler and generally discusses the compilation process.

Chapter 3, "C++ Compiler Options," explains the C++ compiler options in detail and provides task-oriented option groupings.

Chapter 4, "Using Templates," discusses use of templates, including template compilation, definition searching, and instance linkage.

Chapter 5, "Using Libraries," explains how to use the many C++ libraries.

Chapter 6, "Building Libraries," reviews the library-building process.

Multiplatform Release

The Sun™ WorkShop™ C++ compiler documentation applies to the release of the C++ compiler on Solaris 2.5.1, 2.6, and Solaris 7 operating environments on:

- The SPARC™ platform
- The x86 platform, where x86 refers to the Intel implementation of one of the following: Intel 80386, Intel 80486, Pentium, or the equivalent

Note - The latest operating environment release is Solaris 7, but code and path or package path names may use Solaris 2.7 or SunOS 5.7.

Note - The term “x86” refers to the Intel 8086 family of microprocessor chips, including the Pentium, Pentium Pro, and Pentium II processors and compatible microprocessor chips made by AMD and Cyrix. In this document, the term “x86” refers to the overall platform architecture. Features described in this book that are particular to a specific platform are differentiated by the terms “SPARC” and “x86” in the text.

C++ Compiler Related Books

The following books are part of the C++ 5.0 documentation package.

- *C++ Programming Guide* tells you how to use C++ 5.0 features to write more efficient programs. Some of the areas discussed include using templates, exception handling, casting, runtime type identification, and interfacing with FORTRAN 77.
- *C++ Library Reference* describes the C++ libraries, including the C++ Standard Library, the `Tools.h++` Class Library, the Sun WorkShop Memory Monitor, and the `iostream` and complex libraries.
- *C++ Migration Guide* explains what you need to know when moving from 4.0, 4.0.1, 4.1, or 4.2 versions of the C++ compiler to the C++ 5.0 version.
- *Tools.h++ User's Guide* discusses the use of C++ classes for enhancing the efficiency of your programs.
- *Tools.h++ Class Library Reference* provides details on the `Tools.h++` Class Library.
- *C++ Standard Library 2.0 User's Guide* instructs you in the use of the C++ Standard Library, including locales and `iostreams`.
- *C++ Standard Library Class Reference* provides more detailed information on the use of the C++ Standard Library.

- *Sun WorkShop Memory Monitor User's Guide* describes how to use the Sun WorkShop Memory Monitor garbage collection and memory management tools.

Other Sun WorkShop Books

The following books are part of the Sun Visual WorkShop C++ documentation package:

- *Sun WorkShop Quick Install* provides installation instructions.
- *Sun WorkShop Installation and Licensing Reference* provides supporting installation and licensing information.
- *Sun Visual WorkShop C++ Overview* gives a high-level outline of the C++ package suite.
- *Using Sun WorkShop* gives information on performing development operations through Sun WorkShop.
- *C User's Guide* tells how to use the C compiler.
- *Numerical Computation Guide* details floating-point computation numerical accuracy issues.
- *Debugging a Program With dbx* provides information on using dbx commands to debug a program.
- *Analyzing Program Performance With Sun WorkShop* describes the profiling tools; the LoopTool, LoopReport, and LockLint utilities; and use of the Sampling Analyzer to enhance program performance.
- *Sun WorkShop TeamWare User's Guide* describes how to use the Sun WorkShop TeamWare code management tools.
- *Sun WorkShop Performance Library Reference Manual* discusses the library of subroutines and functions to perform useful operations in computational linear algebra and Fourier transforms.
- *Sun WorkShop Visual User's Guide* describes how to use Visual to create C++ and Java™ graphical user interfaces.

Solaris Books

The following Solaris manuals provide additional useful information:

- *The Solaris Linker and Libraries Guide* gives information on linking and libraries.
- The Solaris *Programming Utilities Guide* provides information for developers about the special built-in programming tools available in the SunOS™ system.

Commercially Available Books

The following is a partial list of available books on the C++ language.

Object-Oriented Analysis and Design with Applications, Second Edition, Grady Booch (Addison-Wesley, 1994).

Thinking in C++, Bruce Eckel (Prentice Hall, 1995).

The Annotated C++ Reference Manual, Margaret A. Ellis and Bjarne Stroustrup (Addison-Wesley, 1990).

Design Patterns: Elements of Reusable Object-Oriented Software, Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, (Addison-Wesley, 1995).

C++ Primer, Third Edition, Stanley B. Lippman and Josee Lajoie (Addison-Wesley, 1998).

Effective C++-50 Ways to Improve Your Programs and Designs, Second Edition, Scott Meyers (Addison-Wesley, 1998).

More Effective C++-35 Ways to Improve Your Programs and Designs, Scott Meyers (Addison-Wesley, 1996).

STL Tutorial and Reference Guide-Programming with the Standard Template Library, David R. Musser and Atul Saini (Addison-Wesley, 1996).

C++ for C Programmers, Ira Pohl (Benjamin/Cummings, 1989).

The C++ Programming Language, Third Edition, Bjarne Stroustrup (Addison-Wesley, 1997).

Ordering Sun Documents

The SunDocsSM program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals using this program.

For a list of documents and how to order them, see the catalog section of the SunExpress™ Internet site at <http://www.sun.com/sunexpress>.

Accessing Sun Documents Online

Sun WorkShop documentation is available online from several sources:

- The `docs.sun.com` Web site

- AnswerBook2™ collections
- HTML documents
- Online help and release notes

Using the docs.sun.com Web site

The docs.sun.com Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is <http://docs.sun.com>.

Accessing AnswerBook2 Collections

The Sun WorkShop documentation is also available using AnswerBook2 software. To access the AnswerBook2 collections, your system administrator must have installed the AnswerBook2 documents during the installation process (if the documents are not installed, see your system administrator or Chapter 3 of *Sun WorkShop Quick Install* for installation instructions). For information about accessing AnswerBook2 documents, see Chapter 6 of *Sun WorkShop Quick Install*, Solaris installation documentation, or your system administrator.

Note - To access AnswerBook2 documents, Solaris 2.5.1 users must first download AnswerBook2 documentation server software from a Sun Web page. For more information, see Chapter 6 of *Sun WorkShop Quick Install*.

Accessing HTML Documents

The following Sun Workshop documents are available online only in HTML format:

- Tools.h++ Class Library Reference
- Tools.h++ User's Guide
- *Numerical Computation Guide*
- Standard C++ Library User's Guide
- *Standard C++ Class Library Reference*
- *Sun WorkShop Performance Library Reference Manual*
- *Sun WorkShop Visual User's Guide*
- Sun WorkShop Memory Monitor User's Manual

To access these HTML documents:

1. Open the following file through your HTML browser:

install-directory/SUNWspr0/DOC5.0/lib/locale/C/html/index.html

Replace *install-directory* with the name of the directory where your Sun WorkShop software is installed (the default is /opt).

The browser displays an index of the HTML documents for the Sun WorkShop products that are installed.

2. Open a document in the index by clicking the document's title.

Accessing Sun WorkShop Online Help and Release Notes

This release of Sun WorkShop includes an online help system as well as online manuals. To find out more see:

- **Online Help.** A help system containing extensive task-oriented, context-sensitive help. To access the help, choose Help Help Contents. Help menus are available in all Sun WorkShop windows.
- **Release Notes.** The Release Notes contain general information about Sun WorkShop and specific information about software limitations and bugs. To access the Release Notes, choose Help Release Notes.

Online Books

The docs.sun.com Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is <http://docs.sun.com/>.

The Sun WorkShop documentation is also available using AnswerBook™ software. To access the AnswerBook collections, your system administrator must have installed the AnswerBook package during the installation process. For information on installing and accessing AnswerBook software see the Sun WorkShop installation documentation, the Solaris installation documentation, or your system administrator.

This release of Sun WorkShop includes an online help system as well as online manuals. To find out more, see:

- **Online Help.** A help system containing extensive task-oriented, context-sensitive help. To access the help, choose Help Help Contents. Help menus are available in all Sun WorkShop windows.

- Release Notes. The Release Notes contain general information about Sun WorkShop and specific information about software limitations and bugs. To access the Release Notes, choose Help Release Notes.

What Typographic Changes Mean

The following table describes the typographic changes used in this book.

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output.	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name% You have mail.</code>
AaBbCc123	What you type, contrasted with on-screen computer output.	<code>machine_name% su</code> <code>Password:</code>
<i>AaBbCc123</i>	Command-line placeholder: replace with a real name or value.	To delete a file, type <code>rm filename</code> .
<i>AaBbCc123</i>	Book titles, new words or terms, or words to be emphasized.	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be root to do this.

Compiler options and code samples may use the following conventions:

[]	Square brackets contain arguments that are optional.	<code>-xO[n]</code>
()	Parentheses contain a set of choices for a required option.	<code>-d(y n)</code>
	The “pipe” or “bar” symbol separates arguments, only one of which may be used at one time.	<code>-d(y n)</code>
...	The ellipsis indicates omission in a series.	<code>-xinline=<i>fl</i>[,...<i>fn</i>]</code>

TABLE P-1 Typographic Conventions (continued)

Typeface or Symbol	Meaning	Example
%	The percent sign indicates the word has a special meaning.	-ftrap=%all, no%division
<>	In ASCII files, such as the README file, angle brackets contain a variable that must be replaced by an appropriate value.	-xtemp=<dir>

Shell Prompts in Command Examples

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

TABLE P-2 System Prompts

Shell	Prompt
C shell prompt	machine_name%
C shell superuser prompt	machine_name#
Bourne shell and Korn shell prompt	\$
Bourne shell and Korn shell superuser prompt	#

The C++ Compiler

This chapter provides a brief, conceptual overview of Sun™ C++ and the C++ compiler.

Standards Conformance

The C++ compiler (CC) supports the ISO International Standard for C++, ISO IS 14882:1998, *Programming Language-C++*. The README file that accompanies the current release describes any departures from requirements in the standard

On SPARC platforms, the compiler provides support for the optimization-exploiting features of SPARC V8, and SPARC V9, including the UltraSPARC™ implementation. These features are defined in the SPARC Architecture Manuals, Version 8 (ISBN 0-13-825001-4), and Version 9 (ISBN 0-13-099227-5), published by Prentice-Hall for SPARC International.

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.

Operating Environments

The C++ compiler (CC) integrates with other Sun development tools, such as the Sun WorkShop and the C compiler. The Sun C++ compiler and its runtime library are part of the Sun Visual WorkShop C++. You can use these components to develop threaded applications in multiprocessor Solaris 2.5.1, 2.6, and in Solaris 7 operating environments.

Note - The name of the operating environment is Solaris 7, but code and path or package path names might use Solaris 2.7 or SunOS 5.7. Always follow the code or path as it is written.

Release 5.0 of CC is available in the Solaris 2.5.1, 2.6, and Solaris 7 operating environments on SPARC and x86 devices.

Note - Features that are unique to a particular operating environment or hardware platform are so indicated. However, most aspects of the compilers on these systems are the same, including functionality, behavior, and features. The multiprocessor features are available as part of the Sun WorkShop on the SPARC platform with Solaris 2.5.1, 2.6, and Solaris 7 software, and require a Sun WorkShop license.

See the C++ README files for details.

READMEs

The READMEs directory contains files that describe new features, software incompatibilities, bugs, and information that was discovered after the manuals were printed. The location of this directory depends on where your software was installed.

The READMEs in a standard install would appear in: `/opt/SUNWspro/READMEs/`

The READMEs for all compilers are easily accessed by the `--xhelp=readme` command-line option. For example, `CC -xhelp=readme` displays the C++ README file directly.

Man Pages

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

Sun WorkShop man pages are located in `/opt/SUNWspro/man/` after a standard install of the products. Add this path to your `MANPATH` environment variable to access these Sun WorkShop man pages.

You can display a man page by running the command:

```
demo% 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:

```
demo% man -s 3M ieee_flags
```

See the *C++ Programming Guide* for a complete list of C++ related man pages.

Licensing

The C++ compiler uses network licensing, as described in the *Sun WorkShop Installation Reference*.

If you invoke the compiler, and a license is available, the compiler starts. If no license is available, your request for a license is put in a queue, and your compiler continues when a license becomes available. A single license can be used for any number of simultaneous compiles by a single user on a single machine.

To run C++ and the various utilities, several licenses might be required, depending on the package you have purchased.

New Features of the C++ Compiler

The C++ compiler offers the following new features:

- Compliance with the C++ ISO standard, including:
 - Namespaces and Koenig lookup
 - Type `bool`
 - Array `new` and array `delete`
 - Extended support for templates
 - The C++ standard library
 - Covariant return types on virtual functions
 - Compatibility with C++ 4.0, 4.01, 4.1, and 4.2
- Sun WorkShop Memory Monitor for garbage collection and catching memory leaks
- SPARC V9 support on Solaris 7
- Binary and source compatibility features to aid a smooth transition to ISO C++

The C++ compiler package also includes:

- Online `README` files containing new or changed features, latest known software and documentation bugs, and other late-breaking information
- Man pages that concisely describe a user command or library function
- The C++ name demangling tool set (`dem` and `c++filt`)
- `Tools.h++` class library to simplify your programming

C++ Utilities

The following C++ utilities are now incorporated into traditional UNIX[®] tools and are bundled with the UNIX operating system:

- `lex` - Generates programs used in simple lexical analysis of text
- `yacc` - Generates a C function to parse the input stream according to syntax
- `prof` - Produces an execution profile of modules in a program
- `gprof` - Profiles program run-time performance by procedure
- `tcov` - Profiles program run-time performance by statement

See *Analyzing Program Performance With Sun WorkShop* and associated man pages for further information on these UNIX tools.

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.

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 environment documentation.

Using the C++ Compiler

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`)

Getting Started

This section gives you a brief overview of how to use the C++ compiler to compile and run C++ programs. See Chapter 3 for a full reference to 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:

- Using an editor to create a C++ source file with one of the valid suffixes listed in Table 2-1.
- Invoking the compiler to produce an executable file

- Launching the program into execution by typing the name of the executable file

The following program displays a message on the screen:

```
demo% cat greetings.cc
#include <iostream>
int main() {
    std::cout << ``Real programmers write C++!`` << std::endl;
    return 0;
}
demo% CC greetings.cc
demo% a.out
Real programmers write C++!
demo%
```

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:

```
demo% CC -o greetings greetings.C
```

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:

```
demo% greetings
Real programmers write C++!
demo%
```

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.

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). However, note the following exceptions:

- The above rule does not apply to linker options.
- The `-I`, `-L`, `-pti`, `-R`, and `-xinline` options accumulate, 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:

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

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 2-1 File Name Suffixes Recognized by the C++ Compiler

Suffix	Language	Action
<code>.c</code>	C++	Compile as C++ source files, put object files in current directory; default name of object file is that of the source but with an <code>.o</code> suffix.
<code>.C</code>	C++	Same action as <code>.c</code> suffix.

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

Suffix	Language	Action
.cc	C++	Same action as .c suffix.
.cpp	C++	Same action as .c suffix.
.cxx	C++	Same action as .c suffix.
.i	C++	Preprocessor output file treated as C++ source file. Same action as .C suffix.
.s	Assembler	Assemble source files with the assembler.
.S	Assembler	Assemble source files with both the C language preprocessor and the assembler.
.i1	Inline expansion	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 <i>inline(1)</i> man page.)
.o	Object files	Pass object files through to the linker.
.a	Static (archive) library	Pass object library names to the linker.
.so	Dynamic (shared) library	Pass names of shared objects to the linker.
.so.n		

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. A single source file can contain any number of procedures (main program, function, module, and so on). There are advantages to organizing an application with one procedure per file, as there are for gathering procedures that work together into a single file. Some of these are described in *C++ Programming Guide*.

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`.

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

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.

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:

```
demo% CC -c file1.cc Make new object file
demo% 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).

Consistent Compiling and Linking

If you do compile and link in separate steps, consistent compiling and linking is critical when using the following compiler options:

- `-fast`
- `-g`
- `-g0`
- `-library`
- `-misalign`
- `-mt`
- `-p`
- `-xa`
- `-xarch=a`
- `-xcg92` and `-xcg89`
- `-xpg`
- `-xprofile`
- `-xtarget=t`

If you *compile* any subprogram with any of these options, be sure to *link* with the same option as well:

- In the case of the `-library`, `-fast`, and `-xarch` options, you must be sure to include the linker options that would have been passed if you had compiled and linked together.
- With `-p`, `-xpg` and `-xprofile`, including the option in one phase and leaving it out of the other 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 leaving it out of the other will not affect the correctness of the program, but the program will not be prepared properly for debugging.

In the following example, the programs are compiled using the `-xcg92` compiler option. This option is a macro for `-xtarget=ss1000` and expands to:
`-xarch=v8 -xchip=super -xcache=16/64/4:1024/64/1`

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 last line (the link line) will be used to compile the instantiated templates.

```
demo% CC -c -xcg92 sbr.cc
demo% CC -c -xcg92 smain.cc
demo% CC -xcg92 sbr.o smain.o
```

Compiling for SPARC V9

You can create both 32-bit and 64-bit binaries using the C++ 5.0 compiler.

Compilation, linking, and execution of 64-bit objects can take place only in a V9 SPARC, Solaris 7 environment with a 64-bit kernel running. Compilation for 64-bit Solaris 7 is indicated by the `-xarch=v9` and `-xarch=v9a` options.

Diagnosing the Compiler

You can use the `-verbose` option to display helpful information while compiling a program. See Chapter 3 for more information.

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*, preceded by a dash (-) or a plus sign (+) generate warnings.
- Unrecognized *nonoptions*, 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:

```
demo% 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:

```
demo% CC fast move.cc          <- The user meant to type  
-fast  
move.C:  
ld: fatal: file fast: cannot open file; errno=2  
ld: fatal: File processing errors. No output written to a.out
```

Understanding the Compiler Organization

The C++ compiler package consists of a front end, optimizer, code generator, assembler, template pre-linker, 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.

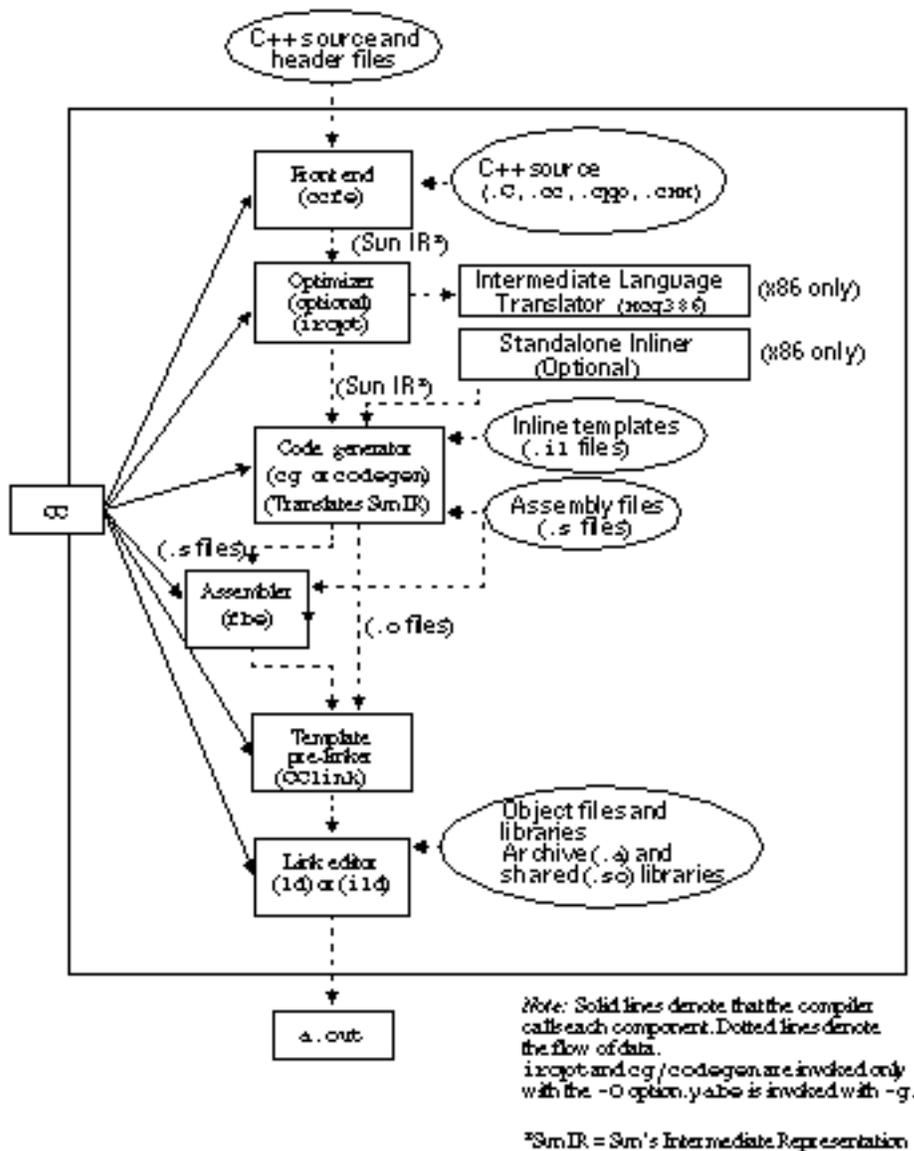


Figure 2-1 The Compilation Process

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

Component	Description	Notes on Use
ccfe	Front end (Compiler preprocessor and compiler)	
iropt	Code optimizer	(SPARC) -xO[2-5], -fast
xcg386	Intermediate language translator	(x86) Always invoked
inline	Inline expansion of assembly language templates	(SPARC) .il file specified
mwinline	Automatic inline expansion of functions	(x86) -xO4, -xinline
fbe	Assembler	
cg	Code generator, inliner, assembler	(SPARC)
codegen	Code generator	(x86)
CCLink	Template pre-linker	
ld	Non-incremental link editor	
ild	Incremental link editor	-g, -xildon

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 `-xOlevel` 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. If the compiler runs out of memory, try reducing the level of optimization. Alternately, split multiple-routine source files into files with one routine per file, using `fsplit(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:

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

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:

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

Control of Virtual Memory

Compiling very large routines (thousands of lines of code in a single procedure) at `-xO3` or higher can require an unreasonable 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 a `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 16 Mbytes:

```
demo$ ulimit -d 16000
```

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

The next example also shows how to limit virtual memory to 16 Mbytes:

```
demo% limit datasize 16M
```

Each of these examples causes the optimizer to try to recover at 16 Mbytes 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 32 Mbytes of swap space, use the following commands:

In a `sh` shell:

```
demo$ ulimit -d 16000
```

In a `cs`h shell:

```
demo% limit datasize 16M
```

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

Memory Requirements

A workstation should have at least 24 megabytes of memory; 32 megabytes are recommended.

To determine the actual real memory, use the following command:

```
demo% /usr/sbin/dmesg | grep mem
mem = 655360K (0x28000000)
avail mem = 602476544
```

Simplifying Commands

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

Using Aliases Within the C Shell

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

```
demo% alias CCfx "CC -fast -xnoibmil"
```

The next example uses the alias `CCfx`.

```
demo% CCfx any.C
```

The command `CCfx` is now the same as:

```
demo% CC -fast -xnoibmil any.C
```

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):

```
demo% setenv CCFLAGS '-silent -fast -Xlist'
```

The next example uses `CCFLAGS` explicitly.

```
demo% 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 -silent -fast -Xlist files...
```

Using make

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

Using CCFLAGS Within make

When you are using the *implicit* compilation rules of the makefile (that is, there is no C++ compile line), CCFLAGS is used automatically by the make program.

Adding a Suffix to Your Makefile

You can incorporate different file suffixes into C++ by adding them to your makefile. The following example adds .C as a valid suffix for C++ files. Add the SUFFIXES macro to your makefile:

```
.SUFFIXES: .cpp .cpp~
```

(This line can be located anywhere in the makefile.)

Add the following lines to your makefile. Indented lines must start with a tab.

```
.cpp:
    $(LINK.cc) -o $@ $< $(LDLIBS)
.cpp~:
    $(GET) $(GFLAGS) -p $< > $*.cpp
    $(LINK.cc) -o $@ $*.cpp $(LDLIBS)
.cpp.o:
    $(COMPILE.cc) $(OUTPUT_OPTION) $<
.cpp~.o:
    $(GET) $(GFLAGS) -p $< > $*.cpp
    $(COMPILE.cc) $(OUTPUT_OPTION) $<
.cpp.a:
    $(COMPILE.cc) -o $% $<
    $(COMPILE.cc) -xar $@ $%
    $(RM) $%
.cpp~.a:
    $(GET) $(GFLAGS) -p $< > $*.cpp
    $(COMPILE.cc) -o $% $<
    $(COMPILE.cc) -xar $@ $%
    $(RM) $%
```

Using make With iostreams

The standard iostream file names do not have “.h” suffixes. Instead, they are named istream, ostream, and so forth. In addition, the template source files are named istream.cc, ostream.cc, and so forth. If you include a standard iostream header, such as <istream>, in your program and your makefile has .KEEP_STATE, you will encounter problems. For example, if you include <istream>, make thinks

that `istream` is an executable and uses the default rules to build `istream` from `istream.cc` resulting in very misleading error messages. (Both `istream` and `istream.cc` are installed under `SC5.0/include/CC`). To prevent `make` from using the default rules, use the `-r` option.

C++ Compiler Options

This chapter details the command-line options for the `CC` compiler running under Solaris 2.5.1, 2.6 and Solaris 7. The features described apply to all platforms except as noted; features unique to one platform are identified as SPARC or x86. See the description of this multiplatform release in the preface.

The following table shows examples of typical option syntax formats.

Syntax Format	Example
<code>-option</code>	<code>-E</code>
<code>-optionvalue</code>	<code>-Ipathname</code>
<code>-option=value</code>	<code>-xunroll=4</code>
<code>-option value</code>	<code>-o filename</code>

The typographical conventions in Table P-1 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.

Some general guidelines for options are:

- The `-llib` option links with library `llib.a` (or `llib.so`). It is always safer to put `-llib` after the file name source and object 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 `-I`, `-L`, `-pti`, `-R`, and `-xinline` options accumulate, not override.

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

Options Summarized by Function

In this section, the compiler options are grouped by function to provide a quick reference.

Code Generation Options

The following code generation options are listed in alphabetical order.

Action	Option	Details
Sets the major release compatibility mode of the compiler.	<code>-compat</code>	" <code>-compat[(4 5)]</code> " on page 41
Does not expand C++ inline functions.	<code>+d</code>	" <code>+d</code> " on page 42
Controls virtual table generation.	<code>+e(0 1)</code>	" <code>+e(0 1)</code> " on page 45
Compiles for use with the debugger.	<code>-g</code>	" <code>-g</code> " on page 58
Produces position-independent code.	<code>-KPIC</code>	" <code>-KPIC</code> " on page 62
Produces position-independent code.	<code>-Kpic</code>	" <code>-Kpic</code> " on page 62
Compiles and links for multithreaded code.	<code>-mt</code>	" <code>-mt</code> " on page 66

Action	Option	Details
Specifies the code address space.	<code>-xcode=a</code>	" <code>-xcode=a</code> " on page 87
Merges the data segment with the text segment.	<code>-xMerge</code>	" <code>-xMerge</code> " on page 94

Debugging Options

The following debugging options are listed in alphabetical order.

Action	Option	Details
Does not expand C++ inline functions.	<code>+d</code>	" <code>+d</code> " on page 42
Shows commands built by the driver, but does not compile.	<code>-dryrun</code>	" <code>-dryrun</code> " on page 44
Runs the preprocessor on source files; does not compile.	<code>-E</code>	" <code>-E</code> " on page 45
Displays a summary list of compiler options.	<code>-flags</code>	" <code>-flags</code> " on page 49
Compiles for use with the debugger.	<code>-g</code>	" <code>-g</code> " on page 58
Compiles for debugging, but doesn't disable inlining.	<code>-g0</code>	" <code>-g0</code> " on page 58
Prints path names of included files.	<code>-H</code>	" <code>-H</code> " on page 59
Displays a summary list of compiler options.	<code>-help</code>	" <code>-help</code> " on page 59

Action	Option	Details
Retains temporary files created during compilation.	-keeptmp	"-keeptmp" on page 61
Explains where to get information about migrating to 5.0	-migration	"-migration" on page 65
Only preprocesses source; outputs to .i file.	-P	"-P" on page 69
Passes an option directly to a compilation phase.	-Qoption	"-Qoption <i>phase option</i> [,... <i>option</i>]" on page 71
Displays the content of the online README file.	-readme	"-readme" on page 73
Strips the symbol table out of the executable file.	-s	"-s" on page 74
Defines directory for temporary files.	-temp= <i>dir</i>	"-temp= <i>dir</i> " on page 75
Controls compiler verbosity.	-verbose= <i>vlst</i>	"-verbose= <i>v</i> [,... <i>v</i>]" on page 77
Turns off the Incremental Linker.	-xildoff	"-xildoff" on page 89
Turns on the Incremental Linker.	-xildon	"-xildon" on page 90
Allows debugging with dbx without object (.o) files.	-xs	"-xs" on page 103

Action	Option	Details
Produces table information for the WorkShop source code browser.	-xsb	"-xsb" on page 104
Produces <i>only</i> source browser information, no compilation.	-xsbfast	"-xsbfast" on page 105

Floating-Point Options

The following floating-point options are listed in alphabetical order.

Action	Option	Details
Disables/enables the SPARC nonstandard floating-point mode.	- fns[=(no yes)]	"-fns[=(yes no)]" on page 50
<i>x86</i> : Sets floating-point precision mode.	- fprecision= <i>p</i>	"-fprecision= <i>p</i> " on page 51
Sets IEEE rounding mode in effect at start-up.	-fround= <i>r</i>	"-fround= <i>r</i> " on page 52
Sets floating-point optimization preferences.	-fsimple= <i>n</i>	"-fsimple[= <i>n</i>]" on page 53
<i>x86</i> : Forces precision of floating-point expressions.	-fstore	"-fstore" on page 55
Sets IEEE trapping mode in effect at start-up.	-ftrap= <i>t</i>	"-ftrap= <i>t</i> " on page 55

Action	Option	Details
<i>x86</i> : Disables forced precision of expression.	<code>-nofstore</code>	" <code>-nofstore</code> " on page 67
Causes <code>libm</code> to return IEEE 754 values for math routines in exceptional cases.	<code>-xlibmieee</code>	" <code>-xlibmieee</code> " on page 91

Language Options

The following language options are listed in alphabetical order.

Action	Option	Details
Sets the major release compatibility mode of the compiler.	<code>-compat</code>	" <code>-compat[=(4 5)]</code> " on page 41
Enables/disables various C++ language features.	<code>-features=<i>alst</i></code>	" <code>-features=<i>a</i>[...<i>a</i>]</code> " on page 47

Library Options

The following library linking options are listed in alphabetical order.

Action	Option	Details
Allows dynamic static library linking.	<code>-B<i>binding</i></code>	" <code>-B<i>binding</i></code> " on page 39
Allows or disallows dynamic libraries for the entire executable.	<code>-d(<i>y n</i>)</code>	" <code>-d(<i>y n</i>)</code> " on page 44
Builds a dynamic shared library instead of an executable file.	<code>-G</code>	" <code>-G</code> " on page 57

Action	Option	Details
Assigns a name to the generated dynamic shared library.	<code>-hname</code>	"-hname" on page 59
Tells <code>ld(1)</code> to ignore any <code>LD_LIBRARY_PATH</code> setting.	<code>-i</code>	"-i" on page 59
Adds <code>dir</code> to the list of directories to be searched for libraries.	<code>-Ldir</code>	"-Ldir" on page 62
Adds <code>liblib.a</code> or <code>liblib.so</code> to the linker's library search list.	<code>-llib</code>	"-llib" on page 62
Forces inclusion of specific libraries and associated files into compilation and linking.	<code>-library=llst</code>	"-library=I[,...I]" on page 63
Compiles and links for multithreaded code.	<code>-mt</code>	"-mt" on page 66
Does not build path for libraries into executable.	<code>-norunpath</code>	"-norunpath" on page 68
Builds dynamic library search paths into the executable file.	<code>-R pathname</code>	"-Rpathname" on page 72
Indicates which C++ libraries are to be linked statically.	<code>-staticlib=llst</code>	"-staticlib=I[,...I]" on page 74
Creates archive libraries.	<code>-xar</code>	"-xar" on page 80
Causes <code>libm</code> to return IEEE 754 values for math routines in exceptional cases.	<code>-xlibmieee</code>	"-xlibmieee" on page 91
Inlines selected <code>libm</code> library routines for optimization.	<code>-xlibmil</code>	"-xlibmil" on page 91

Action	Option	Details
Uses library of optimized math routines.	-xlibmopt	"-xlibmopt" on page 91
Links with the specified, Sun-supplied, licensed libraries.	-xlic_lib= <i>lst</i>	"-xlic_lib= <i>l</i> [... <i>l</i>]" on page 92
Disables linking with default system libraries.	-xnolib	"-xnolib" on page 94
Cancels -xlibmil on the command line.	-xnolibmil	"-xnolibmil" on page 95
Does not use the math routine library.	-xnolibmopt	"-xnolibmopt" on page 96
Forces fatal error if relocations remain against non-writable, allocatable sections.	-ztext	"-ztext" on page 113

Licensing Options

The following licensing options are listed in alphabetical order.

Action	Option	Details
Disables license queueing.	-noqueue	"-noqueue" on page 67
Links with the specified, Sun-supplied, licensed libraries.	-xlic_lib= <i>lst</i>	"-xlic_lib= <i>l</i> [... <i>l</i>]" on page 92
Shows license server information.	-xlicinfo	"-xlicinfo" on page 92

Obsolete Options

Action	Option	Details
See details.	-vdelx	"-vdelx" on page 77

Output Options

The following output options are listed in alphabetical order.

Action	Option	Details
Compiles only; produces object (.o) files, but suppresses linking.	-c	"-c" on page 40
Shows options passed by driver to the compiler, but does not compile.	-dryrun	"-dryrun" on page 44
Runs only preprocessor on C++ source files, and sends result to <code>stdout</code> ; does not compile.	-E	"-E" on page 45
Same as <code>-help</code> .	-flags	"-flags" on page 49
Builds a dynamic shared library instead of an executable file.	-G	"-G" on page 57
Prints path names of included files.	-H	"-H" on page 59
Explains where to get information about migrating to 5.0	-migration	"-migration" on page 65
Sets name of the output or executable file to <i>filename</i> .	-o <i>filename</i>	"-o <i>filename</i> " on page 68
Only preprocesses source; outputs to .i file.	-P	"-P" on page 69
Causes the <code>CC</code> driver to produce output of the type <i>sourcetype</i> .	-Qproduce <i>sourcetype</i>	"-Qproduce <i>sourcetype</i> " on page 72

Action	Option	Details
Displays the contents of the online README file.	-readme	"-readme" on page 73
Strips the symbol table from the executable file.	-s	"-s" on page 74
Controls compiler verbosity.	-verbose= <i>vlst</i>	"-verbose= <i>v[,...v]</i> " on page 77
Prints extra warnings where necessary.	+w	"+w" on page 78
Suppresses warning messages.	-w	"-w" on page 79
Outputs makefile dependency information.	-xM	"-xM" on page 93
Generates dependency information, but excludes /usr/include.	-xM1	"-xM1" on page 94
Produces table information for the Sun WorkShop source code browser.	-xsb	"-xsb" on page 104
Produces <i>only</i> source browser information, no compilation.	-xsbfast	"-xsbfast" on page 105
Reports execution time for each compilation phase.	-xtime	"-xtime" on page 113
Converts all warnings to errors by returning non-zero exit status.	-xwe	"-xwe" on page 113

Performance Options

The following performance options are listed in alphabetical order.

Action	Option	Details
Selects a combination of compilation options for optimum execution speed.	<code>-fast</code>	" <code>-fast</code> " on page 45
Strips the symbol table out of the executable.	<code>-s</code>	" <code>-s</code> " on page 74
Specifies target architecture instruction set.	<code>-xarch=a</code>	" <code>-xarch=a</code> " on page 80
<i>SPARC</i> : Defines target cache properties for the optimizer.	<code>-xcache=c</code>	" <code>-xcache=c</code> " on page 84
Compiles for generic SPARC architecture.	<code>-xcg89</code>	" <code>-xcg(89 92)</code> " on page 85
Compiles for SPARC V8 architecture.	<code>-xcg92</code>	" <code>-xcg(89 92)</code> " on page 85
Specifies target processor chip.	<code>-xchip=c</code>	" <code>-xchip=c</code> " on page 86
Enables linker reordering of functions.	<code>-xF</code>	" <code>-xF</code> " on page 88
Inlines the specified routines to optimize for speed.	<code>-xinline=rst</code>	" <code>-xinline=f[,...f]</code> " on page 90
Inlines selected <code>libm</code> library routines for optimization.	<code>-xlibmil</code>	" <code>-xlibmil</code> " on page 91
<i>SPARC</i> : Uses a library of optimized math routines.	<code>-xlibmopt</code>	" <code>-xlibmopt</code> " on page 91
Cancels <code>-xlibmil</code> on the command line.	<code>-xnolibmil</code>	" <code>-xnolibmil</code> " on page 95

Action	Option	Details
Does not use math library routines.	<code>-xnolibmopt</code>	“ <code>-xnolibmopt</code> ” on page 96
Specifies optimization level to <i>level</i> .	<code>-xOlevel</code>	“ <code>-xO[level]</code> ” on page 96
<i>SPARC</i> : Controls scratch register use.	<code>-xregs=rlist</code>	“ <code>-xregs=r[,...r]</code> ” on page 102
<i>SPARC</i> : Allows no memory-based traps.	<code>-xsafe=mem</code>	“ <code>-xsafe=mem</code> ” on page 104
<i>SPARC</i> : Does not allow optimizations that increase code size.	<code>-xspace</code>	“ <code>-xspace</code> ” on page 105
Specifies a target instruction set and optimization system.	<code>-xtarget=t</code>	“ <code>-xtarget=t</code> ” on page 105
Enables unrolling of loops where possible.	<code>-xunroll=n</code>	“ <code>-xunroll=n</code> ” on page 113

Preprocessor Options

The following preprocessor options are listed in alphabetical order.

Action	Option	Details
Defines symbol <i>name</i> to the preprocessor.	<code>-Dname[=def]</code>	“ <code>-Dname[=def]</code> ” on page 42
Runs preprocessor on C++ source files and sends result to <code>stdout</code> . Does not compile.	<code>-E</code>	“ <code>-E</code> ” on page 45
Only preprocesses source; outputs to <code>.i</code> file.	<code>-P</code>	“ <code>-P</code> ” on page 69

Action	Option	Details
Deletes initial definition of preprocessor symbol <i>name</i> .	<code>-Uname</code>	“ <code>-Uname</code> ” on page 76
Outputs makefile dependency information.	<code>-xM</code>	“ <code>-xM</code> ” on page 93
Generates dependency information, but excludes <code>/usr/include</code> .	<code>-xM1</code>	“ <code>-xM1</code> ” on page 94

Profiling Options

The following profiling options are listed in alphabetical order.

Action	Option	Details
Prepares the object code to collect data for profiling with <code>prof</code> .	<code>-p</code>	“ <code>-p</code> ” on page 69
Generates code for profiling.	<code>-xa</code>	“ <code>-xa</code> ” on page 79
Compiles for profiling with the <code>gprof</code> profiler.	<code>-xpg</code>	“ <code>-xpg</code> ” on page 99
Collects or optimizes with runtime profiling data.	<code>-xprofile=tcov</code>	“ <code>-xprofile=p</code> ” on page 99

Reference Options

The following options provide a quick reference to compiler information.

Action	Option	Details
Displays a summary list of compiler options.	-flags	"-help" on page 59
Displays a summary list of compiler options.	-help	"-help" on page 59
Displays information about migrating to 5.0	-migration	"-migration" on page 65
Displays the contents of the online README file.	-readme	"-readme" on page 73

Source Options

The following source options are listed in alphabetical order.

Action	Option	Details
Adds <i>pathname</i> to the <code>include</code> file search path	-I <i>pathname</i>	"-I <i>pathname</i> " on page 60
Outputs makefile dependency information.	-xM	"-xM" on page 93
Generates dependency information, but excludes <code>/usr/include</code> .	-xM1	"-xM1" on page 94

Template Options

The following template options are listed in alphabetical order.

Action	Option	Details
Controls the placement and linkage of template instances.	<code>-instances=a</code>	" <code>-instances=a</code> " on page 60
Specifies an additional search directory for template source	<code>-ptipath</code>	" <code>-ptipath</code> " on page 70
Specifies the directory of the template repository.	<code>-ptrdirectory</code>	" <code>-ptrdatabase-path</code> " on page 70
Enables/disables various template options.	<code>-template=w</code>	" <code>-template=w</code> " on page 76

Thread Options

The following thread options are listed in alphabetical order.

Action	Option	Details
Compiles and links for multithreaded code.	<code>-mt</code>	" <code>-mt</code> " on page 66
<i>SPARC</i> : Allows no memory-based traps.	<code>-xsafe=mem</code>	" <code>-xsafe=mem</code> " on page 104

How Option Information Is Organized

To facilitate your information search, compiler option information has been 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.

Subsection	Contents
Option Definition	A short definition immediately follows each option. (There is no heading for this category.)
Values	If the option has one or more values, this section defines each value.
Defaults	<p>If the option has a primary or secondary default value, it is stated here.</p> <p>The primary default is the option value in effect if the option is not specified. For example, if <code>-compat</code> is not specified, the default is <code>-compat=5</code>.</p> <p>The secondary default is the option in effect if the option is specified, but no value is given. For example, if <code>-compat</code> is specified without a value, the default is <code>-compat=4</code>.</p>
Expansions	If the option has a macro expansion, it is shown in this section.
Examples	If an example is needed to illustrate the option, it is given here.
Interactions	If the option interacts with other options, the relationship is discussed here. For example, the <code>-xinline</code> option should not be used if <code>-xO</code> is less than 3.
Warnings	If there are cautions regarding use of the option, they are noted here, as are actions that might cause unexpected behavior.
See also	This section contains references to further information in other options or documents.
“Replace With,” “Same as,” or “Use”	<p>If an option has become obsolete and been replaced by another, or the use of one option is preferred over another, the replacement option is noted here. For example, “Replace with <code>-xO</code>” or “Use <code>-xO</code>” means that <code>-xO</code> is the preferred option. Options described this way may not be supported in future releases.</p> <p>If one option is an abbreviation for a longer option, the expansion is shown with “Same as.” If there are two options with the same general meaning and purpose, the preferred option is referenced here. For example, “Same as <code>-xO</code>” indicates that <code>-xO</code> is the preferred option.</p>

Option Reference

The following section alphabetically lists all the C++ compiler options and indicates any platform restrictions.

-386

x86: Use `-xtarget=386`. *This option is provided for backward compatibility only.*

-486

x86: Use `-xtarget=486`. *This option is provided for backward compatibility only.*

-a

Use `-xa`.

-B*binding*

Allows dynamic or requires static library linking.

This option specifies whether the linker should look for dynamic (shared) libraries or for static (archive) libraries. You can use the `-B` option to toggle several times on a command line. This option is passed to the linker, `ld`.

Note - On the Solaris 7 platform, not all libraries available as static libraries.

Values

binding must be one of the following:

Value of <i>binding</i>	Meaning
<code>dynamic</code>	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.
<code>static</code>	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.

(No space is allowed between `-B` and `dynamic` or `static`.)

Defaults

If `-B` is not specified, `-Bdynamic` is assumed.

Interactions

To link the C++ default libraries statically, use the `-staticlib` option.

This option affects the linking of libraries provided by default. To ensure that default libraries are linked dynamically, the last use of `-B` should be `-Bdynamic`.

Examples

The following compiler command links `libfoo.a` even if `libfoo.so` exists; all other libraries are linked dynamically:

```
demo% CC a.o -Bstatic -lfoo -Bdynamic
```

See also

`-nolib`, `-staticlib`, `ld(1)`, “Statically Linking Standard Libraries” on page 130, *Linker and Libraries Guide*

-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.

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.

See also

`-o filename`

`-cg[89 | 92]`

Use `-xcg[89 | 92]`.

`-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.

Values

<code>-compat=4</code>	(Compatibility mode) Set language and binary compatibility to that of the 4.0.1, 4.1, and 4.2 compilers.
<code>-compat=5</code>	(Standard mode) Set language and binary compatibility to that of the 5.0 compiler.

Defaults

If the `-compat` option is not specified, `-compat=5` is assumed.

If only `-compat` is specified, `-compat=4` is assumed.

Regardless of the `-compat` setting, `__SUNPRO_CC` is set to `0x500`.

Interactions

This option controls the preprocessor `__SUNPRO_CC_COMPAT=(4|5)` macro.

The `-xarch=v9` and the `-compat[=4]` options are not supported when used together.

See also

C++ Migration Guide

+d

Does not expand C++ inline functions.

Interactions

This option is automatically turned on when you specify `-g`, the debugging option.

The `-g0` debugging option does not turn on `+d`.

Warnings

For large programs that rely heavily on inline functions, the amount of additional code generated can be substantial.

See also

`-g0`, `-g`

`-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.

Values

The following values are predefined for SPARC and x86:

`__BUILTIN_VA_ARG_INCR` (for the `__builtin_alloca`,
`__builtin_va_alist`, and `__builtin_va_arg_incr` keywords
in `varargs.h`, `stdarg.h`, and `sys/varargs.h`)

`__cplusplus`

`__DATE__`

`__FILE__`

`__LINE__`

`__STDC__`

`__TIME__`

`__sun`

The following values are predefined for sun

`__SUNPRO_CC=0x500` (The value of `__SUNPRO_CC` indicates the release number
of the compiler)

`__SUNPRO_CC_COMPAT=(4|5)`

`__SVR4`

`__'uname -s' 'uname -r'` (Where `uname` is the output of `uname -s` with invalid
characters replaced by underscores, as in `-D__SunOS_5_3`; `-D__SunOS_5_4`)

`__unix`

The following values are predefined for UNIX

`_WCHAR_T`

The following values are predefined for SPARC only:

`__sparc` (32-bit compilation modes only)

The following values are predefined for SPARC v9 only:

`__sparcv9` (64-bit compilation modes only)

The following values are predefined for x86 only:

`__i386`

You can use these values in such preprocessor conditionals as `#ifdef`.

Defaults

If you do not use `=def`, `name` is defined as 1.

See also

-U

`-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.

Values

<code>-dy</code>	Specifies dynamic linking in the link editor.
<code>-dn</code>	Specifies static linking in the link editor.

Defaults

If no `-d` option is specified, `-dy` is assumed.

See also

`ld(1)`, *Linker and Libraries Guide*

`-dalign`

SPARC: Generates `double-word` load and store instructions whenever possible for improved performance.

This option assumes that all `double` type data are `double-word` aligned.

Warnings

If you compile one program unit with `-dalign`, compile all units of a program with `-dalign`, or you might get unexpected results.

`-dryrun`

Shows commands built by driver, but does not compile.

This option directs the driver `CC` to show, but not execute, the subcommands constructed by the compilation driver.

-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.

Output from this option is not supported as input to the C++ compiler when templates are used.

See also

-P

+e(0|1)

Controls virtual table generation when `-compat=4`. Invalid and ignored when `-compat=5`.

Values

+e0	Suppresses the generation of virtual tables and creates external references to those that are needed.
+e1	Creates virtual tables for all defined classes with virtual functions.

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.

See also

C++ Migration Guide

-fast

Optimizes for speed of execution using a selection of options.

This option is a macro that selects a combination of compilation options for optimum execution speed.

The criteria for the `-fast` option vary with the C, C++, FORTRAN 77, and Pascal compilers. See the appropriate compiler documentation for specifics.

Expansions

This option provides near maximum performance for many applications by expanding to the following compilation options:

Option	SPARC	x86
<code>-dalign</code>	X	-
<code>-fns</code>	X	-
<code>-fsimple</code>	X	-
<code>-ftrap=%none</code>	X	X
<code>-nofstore</code>	-	X
<code>-xlibmil</code>	X	X
<code>-xlibmopt</code>	X	X
<code>-xO4</code>	X	X
<code>-xtarget=native</code>	X	X

Interactions

The `-fast` option expands into options that may affect options that are specified after `-fast`.

The code generation option, optimization level, and use of inline template files can be overridden by subsequent options. 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`:

```
demo% CC -fast -xO3
```

The following compiler command results in an optimization level of `-xO4`:

```
demo% CC -xO3 -fast
```

Warnings

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.

Note - In previous SPARC releases, the `-fast` macro option included `-fnonstd`; now it does not. Nonstandard floating-point mode is not initialized by `-fast`. See the *Numerical Computation Guide*, `ieee_sun(3M)`.

See also

`-dalign`, `-fns`, `-fsimple`, `-ftrap=%none`, `-libmil`, `-nofstore`, `-xO4`,
`-xlibmopt`, `-xtarget=native`

`-features=a[,...a]`

Enables/disables various C++ language features named in a comma-separated list.

Values

For `compat=4` only, `a` can be one of the following values:

Value of <code>a</code>	Meaning
<code>[no%]namespace</code>	[Do not] Recognize the keywords <code>namespace</code> and <code>using</code> .
<code>[no%]rtti</code>	[Do not] Allow runtime type information (RTTI).

For `compat=4` and `compat=5`, `a` can be one of the following values:

Value of a	Meaning
[no%]altspell	[Do not] Recognize alternative token spellings (for example, “and” for “&&”).
[no%]arraynew	[Do not] Recognize array forms of operator new and operator delete (for example, operator new [](void*)). When enabled, the macro <code>_ARRAYNEW=1</code> . When not enabled, the macro is not defined.
[no%]anachronisms	[Do not] Allow anachronistic constructs.
[no%]bool	[Do not] Allow the bool type and literals.
[no%]conststrings	[Do not] Put literal strings in read-only memory.
[no%]except	[Do not] Allow C++ exceptions.
[no%]explicit	[Do not] Recognize the keyword explicit.
[no%]export	[Do not] Recognize the keyword export.
[no%]export	[Do not] Allow a \$ as a non-initial identifier character.
[no%]iddollar	[Do not] Use new local-scope rules for the for statement.
[no%]localfor	[Do not] Recognize the keyword mutable.
[no%]mutable	

Note - [no%]castop is allowed for compatibility with makefiles written for the C++ 4.2 compiler, but has no affect on the 5.0 compiler. The new style casts (`const_cast`, `dynamic_cast`, `reinterpret_cast`, and `static_cast`) are always recognized and cannot be disabled.

When `-features=arraynew` is enabled, the macro `_ARRAYNEW=1` is defined; when disabled, the macro is not defined.

When `-features=bool` is enabled, the macro `_BOOL=1` is defined; when disabled, the macro is not defined.

When `-features=no%anachronisms` is enabled, no anachronistic constructs are allowed.

When `-features=except` is disabled, a throw-specification on a function is accepted but ignored. The keywords `try`, `throw`, and `catch` are always reserved.

Defaults

If `-features` is not specified, the following is assumed:

- **Compatibility mode** (`compat=4` or `compat`):
`-features=%none,anachronisms,no%conststrings,except`
- **Standard mode** (`compat=5` or no `compat` option specified):
`-features=%all,conststrings,no%iddollar`

See also

C++ Migration Guide

`-flags`

Same as `-help`.

The `-flags` option displays a brief description of each compiler option.

See also

`-help`, `-readme`, `-migration`.

`-fnonstd`

x86: Causes nonstandard initialization of floating-point hardware.

In addition, the `-fnonstd` option 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).

Defaults

If `-fnonstd` is not specified, IEEE 754 floating-point arithmetic exceptions do not abort the program, and underflows are gradual.

See also

`-fns`, `-ftrap=common`, *Numerical Computation Guide*.

`-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`. (See “Examples.”)

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.

Values

<code>yes</code>	Selects nonstandard floating-point mode
<code>no</code>	Selects standard floating-point mode

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

To use the `-fns` option following some other macro option that includes `-fns`, such as `-fast`:

```
demo% 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:

Note: Nonstandard floating-point mode enabled See the Numerical Computation Guide, ieee

See also

Numerical Computation Guide

`-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.

Values

p must be one of the following:

Value of <i>p</i>	Meaning
<code>single</code>	Rounds to an IEEE single-precision value
<code>double</code>	Rounds to an IEEE double-precision value
<code>extended</code>	Rounds to the maximum precision available

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.

`-fround=r`

Sets the IEEE rounding mode in effect at start-up.

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.

Values

`r` must be one of the following:

Value of <code>r</code>	Meaning
<code>nearest</code>	Rounds towards the nearest number and breaks ties to even numbers.
<code>tozero</code>	Rounds to zero.

Value of <i>r</i>	Meaning
negative	Rounds to negative infinity.
positive	Rounds to positive infinity.

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.

`-fsimple[=n]`

Selects floating-point optimization preferences.

This option allows the optimizer to make simplifying assumptions concerning floating-point arithmetic.

Values

If *n* is present, it must be 0, 1, or 2.

Permit no simplifying assumptions. Preserve strict IEEE 754 conformance.

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.

Permit 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.

Defaults

If `-fsimple` is not designated, the compiler uses `-fsimple=0`.

If `-fsimple` is designated but no value is given for n , the compiler uses `-fsimple=1`.

Interactions

`-fast` implies `-fsimple=1`.

Warnings

This option can break IEEE 754 conformance.

See also

`-fast`

`-fstore`

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 `-nofstore` option.

Warnings

Due to roundoffs and truncation, the results can be different from those that are generated from the register values.

See also

`-nofstore`

`-ftrap=t`

Sets the IEEE trapping mode in effect at start-up.

This option sets the IEEE 754 trapping modes that are established at program initialization, but does not install a `SIGFPE` handler. You can use `ieee_handler` to simultaneously enable traps and install a `SIGFPE` handler. When more than one value is used, the list is processed sequentially from left to right.

Values

t is a comma-separated list that consists of one or more of the following:

Value of <i>t</i>	Meaning
<code>[no%]division</code>	[Do not] Trap on division by zero.
<code>[no%]inexact</code>	[Do not] Trap on inexact result.
<code>[no%]invalid</code>	[Do not] Trap on invalid operation

Value of <i>t</i>	Meaning
[no%]overflow	[Do not] Trap on overflow.
[no%]underflow	[Do not] Trap on underflow.
%all	Trap on all of the above.
%none	Trap on none of the above.
common	Trap on invalid, division by zero, and overflow.

Note that the [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 [no%] form of the option by itself does not explicitly cause a particular trap to be disabled.

If you wish to enable the IEEE traps, `-ftrap=common` is the recommended setting.

Defaults

If `-ftrap` is not specified, the `-ftrap=%none` value is assumed. (Traps are not enabled automatically.)

Examples

When one or more terms are given, the list is processed sequentially from left to right, thus `-ftrap=%all,no%inexact` means to set all traps except `inexact`.

Interactions

The mode can be changed at runtime with `ieee_handler(3M)`.

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=common`.

See also

`ieee_handler(3M)` man page

-G

Build a dynamic shared library instead of an executable file.

All source files specified in the command line are compiled with `-Kpic` 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.

Interactions

The following options are passed to `ld` 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++.

See also

`-dy`, `-Kpic`, `-xcode=pic13`, `-xildoff`, `-ztext`, `ld(1)` man page, *C++ Library Reference*.

`-g`

Instructs both the compiler and the linker to prepare the file or program for debugging.

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

Interactions

You can use this option with `-xO` to get the optimization level that you desire.

If you use this option with `-xO`, you will get limited debugging information.

When you specify this option, the `+d` option is specified automatically.

This option makes `-xildon` the default incremental linker option in order to speed up the compile-edit-debug cycle.

This option invokes `ild` in place of `ld` unless any of the following are true:

- The `-G` option is present
- The `-xildoff` option is present
- Any source files are named on the command line

See also

`+d`, `-g0`, `-xildoff`, `-xildon`, `-xs`, `ld(1)` man page, *Debugging a Program With dbx* (for details about stabs)

`-g0`

Compiles and links for debugging, but doesn't disable inlining.

This option is the same as `-g`, except that `+d` is disabled.

See also

`+d`, `-g`, `-xildon`, *Debugging a Program With dbx*

`-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.

`-help`

Displays a summary list of compiler options.

This option displays a brief description of each compiler option.

See also

`-flags`, `-readme`, *C++ Migration Guide*

`-hname`

Assigns the name *name* to the generated dynamic shared library.

This is a loader 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 `-hname` 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.

Examples

```
demo% CC -G -o libx.so.1 -h libx.so.1 a.o b.o c.o
```

`-i`

Tells the linker, `ld`, to ignore any `LD_LIBRARY_PATH` setting.

-I*pathname*

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 preprocessor searches for #include files in the following order:

1. For includes of the form #include "foo.h" (where quotation marks are used), the directory containing the source file is searched
2. For includes of the form #include <foo.h> (where angle brackets are used), the directory containing the source file is *not* searched
3. The directories named with -I options, if any
4. The standard directory for C++ header files at the following default directory:
/opt/SUNWspro/SC5.0/include/CC
5. Special-purpose files: /opt/SUNWspro/SC5.0/include/cc
6. In /usr/include

Interactions

If `-ptipath` is not used, the compiler looks for template files in `-Ipathname`.

Use `-Ipathname` instead of `-ptipath`.

-inline=*rlst*

Use `-xinline=rlst`.

-instances=*a*

Controls the placement and linkage of template instances.

Values

a must be one of the following:

Value of <code>a</code>	Meaning
<code>extern</code>	Places all needed instances into the template repository and gives them global linkage. (If an instance in the repository is out of date, it is reinstantiated.)
<code>explicit</code>	Places explicitly instantiated instances into the current object file and gives them global linkage. Does not generate any other needed instances.
<code>global</code>	Places all needed instances into the current object file and gives them global linkage.
<code>semiexplicit</code>	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 static linkage. Does not generate any other needed instances.
<code>static</code>	Places all needed instances into the current object file and gives them static linkage.

Defaults

If `-instances` is not specified, `-instances=extern` is assumed.

Warnings

For all values other than `extern`, the template definitions must be included (directly or indirectly) within the current compilation unit. The compiler silently skips the instantiation if the definition is not available.

See also

Chapter 4

`-keeptmp`

Retains temporary files created during compilation.

Along with `-verbose=diags`, this option is useful for debugging.

See also

`-v`, `-verbose`

`-KPIC`

SPARC: Same as `-xcode=pic32`.

x86: Same as `-Kpic`.

`-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.

`-Ldir`

Adds *dir* to list of directories to search for libraries.

This option is passed to `ld`. The directory, *dir*, is searched before compiler-provided directories.

`-llib`

Adds library `llib.a` or `llib.so` to the linker's list of search libraries.

This option is passed to `ld`. Normal libraries have names such as `llib.a` or `llib.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`.

Use this option after your object file name.

Interactions

It is always safer to put `-lx` after the list of sources and objects to insure that libraries are searched in the correct order.

See also

`-Ldir`, *C++ Library Reference*, and *Tools.h++ Class Library Reference*

`-libmieee`

Use `-xlibmieee`.

`-libmil`

Use `-xlibmil`.

`-library=I[,...I]`

Incorporates specified CC-provided libraries into compilation and linking.

Values

For `-compat=4`, *I* must be one of the following:

Value of <i>I</i>	Meaning
<code>[no%]rwttools7</code>	[Do not] Use <code>Tools.h++ v 7</code>
<code>[no%]rwttools7_dbg</code>	[Do not] Use debug-enabled <code>Tools.h++ v 7</code>
<code>[no%]complex</code>	[Do not] Use <code>libcomplex</code> for complex arithmetic
<code>[no%]libc</code>	[Do not] Use <code>libc</code> , the C++ support library
<code>[no%]gc</code>	[Do not] Use <code>libgc</code> , garbage collection
<code>[no%]gc_dbg</code>	[Do not] Use debug-enabled <code>libgc</code> , garbage collection
<code>%all</code>	The same as <code>%none</code> , <code>rwttools7</code> , <code>complex</code> , <code>gc</code> , <code>libc</code>
<code>%none</code>	Use no C++ libraries

For `-compat=5`, `l` must be one of the following:

Value of <code>l</code>	Meaning
<code>[no%]rwtools7</code>	[Do not] Use <code>Tools.h++ v 7</code>
<code>[no%]rwtools7_dbg</code>	[Do not] Use debug-enabled <code>Tools.h++ v 7</code>
<code>[no%]iostream</code>	[Do not] Use <code>libiostream</code> , the classic <code>iostreams</code> library
<code>[no%]Cstd</code>	[Do not] Use <code>libCstd</code> , the C++ standard library
<code>[no%]Crun</code>	[Do not] Use <code>libCrun</code> , the C++ runtime library
<code>[no%]gc</code>	[Do not] Use <code>libgc</code> , garbage collection
<code>[no%]gc_dbg</code>	[Do not] Use debug-enabled <code>libgc</code> , garbage collection
<code>%all</code>	Same as <code>%none</code> , <code>rwtools7</code> , <code>gc</code> , <code>iostream</code> , <code>Cstd</code> , <code>Crun</code>
<code>%none</code>	Use no C++ libraries, except for <code>libCrun</code>

Defaults

For `-compat=4`, if `-library` is not specified, `-library=%none,libC`. The `libC` library always is included unless specifically excluded using `-library=%none` or `-library=no%libC`.

For `-compat=5`, if `-library` is not specified, `-library=%none,Cstd,Crun`. The `libCstd` library always is included unless specifically excluded using `-library=%none` or `-library=no%Cstd`. The `libCrun` library always is included unless specifically excluded using `-library=no%Crun`.

Examples

To link in standard mode without any C++ libraries (except `libCrun`), use:

```
demo% CC -library=%none
```

To include the Rogue Wave `tools.h++ v7` and `iostream` libraries in standard mode:

```
demo% CC -library=rwtools7,iostream
```

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.

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 system support libraries.

Only one Rogue Wave tools library can be used at a time.

The Rogue Wave `Tools.h++` version 7 library is built with classic iostreams. Therefore, when you include the Rogue Wave tools library in standard mode, you must also include `libiostream`.

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.

Programs linking neither `libC` nor `libCrun` might not use all features of the C++ language.

If `-xnolib` is specified, `-library` is ignored.

Warnings

The set of libraries is not stable and might change from release to release.

See also

`-I`, `-l`, `-R`, `-staticlib`, `-xnolib`, “Using make With iostreams” on page 20, *C++ Library Reference*, *Tools.h++ User’s Guide*, *Tools.h++ Class Library Reference*, *Standard C++ Library Reference*

`-migration`

Explains where to get information about migrating source code to C++ 5.0.

See Also

C++ Migration Guide: C++ 4.2 to C++ 5.0 contains information about incompatibilities between versions 4.0.1, 4.1, and 4.2 of the compiler and the 5.0 compiler.

-misalign

SPARC: Permits misaligned data, which would otherwise generate an error, in memory. This is shown in the following code:

```
char b[100];
int f(int * ar) {
return *(int *) (b +2) + *ar;
}
```

Very conservative loads and stores must be used for the data, that is, one byte at a time. Using this option can cause significant degradation in performance when you run the program.

Warnings

If possible, do not link aligned and misaligned parts of the program.

-mt

Compiles and links for multithreaded code.

This option compiles source files with `-D_REENTRANT` and augments the set of support libraries to include `-pthread` in the required order.

Warnings

To ensure proper library linking order, use this option, rather than `-pthread`, to link with `libpthread`.

If you compile and link in separate steps and you compile with `-mt`, be sure to link with `-mt`, as shown in the following example, or you might get unexpected results.

```
demo% CC -c -mt myprog.cc
demo% CC -mt myprog.o
```

See also

`-xnoolib`, *Multithreaded Programming Guide*, *Linker and Libraries Guide*, *C++ Library Reference*

`-native`

Use `-xtarget=native`.

`-noex`

Use `-features=no%except`.

`-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.
- The expression or function is cast to a shorter floating-point type.

See also

`-fstore`

`-nolib`

Use `-xnolib`.

`-nolibmil`

Use `-xnolibmil`.

`-noqueue`

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.

`-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 helpful if you have installed the compiler in some nonstandard location, and you ship an executable file to your customers. Your customers don't have to work with that nonstandard location.

Interactions

If you use any shared libraries under the compiler installed area (default location `/opt/SUNWsprow/lib`) 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.

`-O`

Same as `-xO2`.

`-Olevel`

Use `-xOlevel`.

`-o filename`

Sets the name of the output file or the executable file to *filename*.

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.

`+p`

Disallows preprocessor asserts.

Defaults

If `+p` is not present, preprocessor asserts are allowed.

`-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.

See also

`-E`

`-p`

Prepares object code to collect data for profiling with `prof`.

This option invokes a runtime recording mechanism that produces a `mon.out` file at normal termination.

See also

`-xpg`, *analyzer(1)* man page.

`-pentium`

x86: Replace with `-xtarget=pentium`.

`-pg`

Use `-xpg`.

-PIC

Use `-KPIC`.

-pic

Use `-Kpic`.

-pta

Use `-template=wholeclass`.

-pti*path*

Specifies an additional search directory for template source.

This option is an alternative to the normal search path set by `-Ipathname`. If the `-ptipath` option is used, the compiler looks for template definition files on this path and ignores the `-Ipathname` option.

Using the `-Ipathname` option instead of `-ptipath` produces less confusion.

See also

`-Ipathname`

-pto

Use `-instances=static`.

-ptr*database-path*

Specifies the directory of the template repository.

The template repository is contained within the `SunWS_config` or `SunWS_cache` subdirectory of the given directory. The template repository cache files are stored in `directory/SunWS_cache`. The template repository configuration files are stored in `directory/SunWS_config`.

You cannot use multiple `-ptr` options.

Examples

`-ptr/tmp/Foo` specifies the repository subdirectories
`/tmp/Foo/SunWS_cache` and `/tmp/Foo/SunWS_config`.

Interactions

The subdirectory names can be changed with the environment variables
`SUNWS_CACHE_NAME` and `SUNWS_CONFIG_NAME`.

Warnings

If you compile and link in separate steps, and use `-ptr` to compile, you must also use `-ptr` to link.

`-ptv`

Use `-verbose=template`.

`-Qoption phase option[,...option]`

Passes *option* to the compilation *phase*.

To pass multiple options, specify them in order as a comma-separated list.

Values

phase must have one of the following values:

SPARC Architecture	x86 Architecture
<code>ccfe</code>	<code>ccfe</code>
<code>iropt</code>	<code>cg386</code>
<code>cg</code>	<code>codegen</code>
<code>Cclink</code>	<code>Cclink</code>
<code>ld</code>	<code>ld</code>

Examples

In the following command line, when `ld` is invoked by the `CC` driver, `-Qoption` passes the `-i` and `-m` options to `ld`:

```
demo% CC -Qoption ld -i,-m test.c
```

`-qoption` *phase option*

Use `-Qoption`.

`-qp`

Same as `-p`.

`-Qproduce` *sourcetype*

Causes the `CC` driver to produce output of the type *sourcetype*.

Sourcetype suffixes are defined below.

Suffix	Meaning
<code>.i</code>	Preprocessed C++ source from <code>ccfe</code>
<code>.o</code>	Object file from <code>cg</code> , the code generator
<code>.s</code>	Assembler source from <code>cg</code>

`-qproduce` *sourcetype*

Use `-Qproduce`.

`-Rpathname`

Builds dynamic library search paths into the executable file.

Multiple instances of `-Rpathname` are concatenated, with each *pathname* separated by a colon.

This option is passed to `ld`.

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).

<code>-xarch</code> Value	Default Library Search Path
v9 or v9a	<code>install_dir/SUNWspro/lib/v9</code>
All other values	<code>install_dir/SUNWspro/lib</code>

With a standard install, *install-path* is `/opt`.

Interactions

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*

`-readme`

Displays the content of the `README` file.

The `README` file is paged by the command specified in the environment variable, `PAGER`. If `PAGER` is not set, the default paging command is `more`.

-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.

-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`.

-sb

Replace with `-xsb`.

-sbfast

Use `-xsbfast`.

-staticlib=*l*[,*...*]

Indicates which C++ libraries are to be linked statically.

Values

l must be one of the following:

Value of <i>l</i>	Meaning
<code>[no%]<i>library</i></code>	See <code>-library</code> for the allowable values for <i>library</i> .
<code>%all</code>	All libraries specified in the <code>-library</code> option are linked statically.
<code>%none</code>	Link no libraries specified in the <code>-library</code> option, statically.

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`:

```
demo% CC -staticlib=Crun
```

However, the following command line does not link `libgc` because `libgc` is not linked unless explicitly specified with the `-library` option:

```
demo% CC -staticlib=gc
```

To link `libgc` statically, use the following command:

```
demo% CC -library=gc -staticlib=gc
```

Interactions

If you select C++ libraries explicitly with the `-library` option or implicitly through its defaults, you must also specify `-staticlib` to link these libraries statically. If a you specify a library with `-staticlib` but have not selected it with `-library` or its defaults, the library is not linked. Refer to the examples.

In the Solaris 7 operating environment, some C++ libraries are not available as static libraries.

Warnings

This set of libraries is not stable and might change from release to release.

See also

`-library`, “Statically Linking Standard Libraries” on page 130

`-temp=dir`

Defines directory for temporary files.

This option sets the name of the directory for temporary files, generated during the compilation process, to *dir*.

See also

-keeptmp

-template=*w*

Enables/disables various template options.

Values

w must be wholeclass or no%wholeclass.

[no%]wholeclass	Does [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.
-----------------	---

-time

Use -xtime.

-U*name*

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. It has no effect on any other predefined macros, nor any macro definitions in source files.

You can specify multiple -U options on the command line.

Warnings

This option is order-sensitive if used with the -D option, and is processed after all -D options are processed.

`-unroll=n`

Same as `-xunroll=n`.

`-V`

Same as `-verbose=version`.

`-v`

Same as `-verbose=diags`.

`-vdelx`

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.

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. This option is available only for `-compat=4`. Don't use this option unless you have bought some software from a third-party vendor and the vendor recommends using this option.

`-verbose=v[,...v]`

Controls compiler verbosity.

Values

`v` must be one of the following values.

Value of <code>v</code>	Meaning
<code>[no%]template</code>	[Do not] Turn on the template instantiation <code>verbose</code> mode (sometimes called the “verify” mode). The <code>verbose</code> mode displays each phase of instantiation as it occurs during compilation. Use this option if you are new to templates.
<code>[no%]diags</code>	[Do not] Print the command line for each compilation pass.
<code>[no%]version</code>	[Do not] Direct the <code>CC</code> driver to print the names and version numbers of the programs it invokes.
<code>%all</code>	Invokes all of the above.
<code>%none</code>	Invokes none of the above.

You can specify more than one option, for example, `-verbose=template,diags`.

Defaults

If `-verbose` is not specified, `-verbose=%none` is assumed.

`+W`

Prints extra warnings where necessary.

This option generates additional warnings about questionable constructs that are:

- Nonportable
- Likely to be mistakes
- Inefficient

Defaults

If `+w` is not specified, the compiler warns about constructs that are almost certainly problems.

See also

`-w`, `+w2`

`+w2`

Prints even more warnings.

See also

`+w`

`-W`

Suppresses warning messages.

This option causes the compiler *not* to print warning messages. However, some warnings, particularly warnings regarding serious anachronisms cannot be suppressed.

See also

`+w`

`-xa`

Generates code for profiling.

If set at compile time, the `TCOVDIR` 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.

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 others that have been compiled with `-xa`. You cannot compile a single file with both options.

`-xa` 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, *Analyzing Program Performance With Sun WorkShop*

`-xar`

Creates archive libraries.

When building a C++ archive that uses templates, it is necessary in most cases to include in the archive those template functions that are instantiated in the template database. Using this option automatically adds those templates to the archive as needed.

Examples

The following command line archives the template functions contained in the library and object files.

```
demo% CC -xar -o libmain.a a.o b.o c.o
```

Warnings

Do not add `.o` files from the template data base 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

Chapter 6

`-xarch=a`

Specifies the target architecture instruction set.

Although this option can be used alone, its primary use is to override a value supplied by the `-xtarget` option.

This option limits the instructions generated to those appropriate to the specified architecture, and it *allows* the specified set of instructions. The option does not guarantee an instruction will be used; however, under optimization, it is usually used.

Values

a must be one of the following:

Platform	Value of <i>a</i>
SPARC	<code>generic</code> , <code>v7</code> , <code>v8a</code> , <code>v8</code> , <code>v8plus</code> , <code>v8plusa</code> , <code>v9</code> , <code>v9a</code>
x86	<code>generic</code> , <code>386</code> , <code>486</code> , <code>pentium</code> , <code>pentium_pro</code>

For SPARC

SPARC architectures `v7`, `v8a`, and `v8` are all binary compatible. `v8plus` and `v8plusa` are binary compatible with each other, and they are forward, but not backward, compatible.

For any particular choice, the generated executable might run much more slowly on earlier architectures.

Value of <i>a</i>	Meaning
<code>generic</code>	Gets good performance on most SPARC processors, major degradation on none. With each new release, this best instruction set will be adjusted, if appropriate.
<code>v7</code>	Limits instruction set to V7 architecture. This option uses the best instruction set for good performance on the V7 architecture, but without the quad-precision floating-point instructions. This is equivalent to using the best instruction set for good performance on the V8 architecture, but without the following instructions: <ul style="list-style-type: none"> ■ The quad-precision floating-point instructions ■ The integer <code>mul</code> and <code>div</code> instructions ■ The <code>fsmuld</code> instruction Examples: SPARCstation™ 1, SPARCstation 2

Value of a	Meaning
v8a	<p>Limits instruction set to the V8a version of the V8 architecture.</p> <p>By definition, V8a means the V8 architecture, but without:</p> <ul style="list-style-type: none"> ■ The quad-precision floating-point instructions ■ The <code>fsmuld</code> instruction <p>Example: Any machine based on microSPARC™ I chip architecture</p>
v8	<p>Limits instruction set to V8 architecture.</p> <p>This option uses the best instruction set for good performance on the V8 architecture, but without quad-precision floating-point instructions.</p> <p>Example: SPARCstation 10</p>
v8plus	<p>Limits instruction set to the V8plus version of the V9 architecture.</p> <p>By definition, V8plus means the V9 architecture, but:</p> <ul style="list-style-type: none"> ■ Without the quad-precision floating-point instructions ■ Limited to the 32-bit subset defined by the V8+ specification ■ Without the VIS instructions <p>In V8plus, a system with the 64-bit registers of V9 runs in 32-bit addressing mode, but the upper 32 bits of the <code>i</code> and <code>l</code> registers must not affect program results.</p> <p>Example: Any machine based on UltraSPARC™ chip architecture.</p> <p>Use of this option also causes the <code>.o</code> file to be marked as a V8plus binary. Such files do not run on a V7 or V8 machine.</p>
v8plusa	<p>Limits instruction set to the V8plusa architecture variation.</p> <p>By definition, V8plusa means the V8plus architecture, plus:</p> <ul style="list-style-type: none"> ■ The UltraSPARC specific instructions ■ The VIS instructions <p>This option uses the best instruction set for good performance on the UltraSPARC architecture, but limited to the 32-bit subset defined by the V8plus specification.</p> <p>Example: Any machine based on UltraSPARC chip architecture</p> <p>Use of this option also causes the <code>.o</code> file to be marked as a Sun-specific V8plus binary. Such files do not run on a V7 or V8 machine.</p>

Value of <i>a</i>	Meaning
v9	<p>Limits instruction set to the SPARC-V9 architecture.</p> <p>The resulting <code>.o</code> object files are in 64-bit ELF format and can only be linked with other object files in the same format.</p> <p>The resulting executable can be run only on a 64-bit SPARC processor running 64-bit Solaris 7 software with the 64-bit kernel.</p> <p>Compiling with this option uses the best instruction set for good performance on the V9 SPARC architecture, but without the use of quad-precision floating-point instructions (available only with 64-bit Solaris 7 software).</p>
v9a	<p>Limits instruction set to the SPARC-V9 architecture, adding the Visual Instruction Set (VIS) and extensions specific to UltraSPARC processors. The resulting <code>.o</code> object files are in 64-bit ELF format and can be linked only with other object files in the same format.</p> <p>The resulting executable can be run only on a 64-bit UltraSPARC processor running 64-bit Solaris 7 software with the 64-bit kernel.</p> <p>Compiling with this option uses the best instruction set for good performance on the V9 UltraSPARC architecture, but without the use of quad-precision floating-point instruction (available only with 64-bit Solaris 7 software).</p>

For x86

`generic` and `386` are equivalent in this release.

`486`, `pentium`, or `pentium_pro` directs the compiler to issue instructions for the Intel PentiumPro chip.

Defaults

If `-xarch=a` is not specified, `-xarch=generic` is assumed.

Interactions

The `-xarch=v9` option and the `-compat[=4]` options are not supported when used together.

Warnings

If this option is used 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.

`-xcache=c`

SPARC: Defines cache properties for use by the optimizer.

This option specifies the cache properties that the optimizer can use. It 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.

Values

Value of <i>c</i>	Meaning
<code>generic</code>	Defines the cache properties for good performance on most SPARC processors
<code>s1/l1/a1</code>	Defines level 1 cache properties
<code>s1/l1/a1:s2/l2/a2</code>	Defines level 1 and 2 cache properties.
<code>s1/l1/a1:s2/l2/a2:s3/l3/a3</code>	Defines level 1, 2, and 3 cache properties

The definitions of the cache properties, *si/li/ai*, are as follows:

<i>si</i>	The <i>size</i> of the data cache at level <i>i</i> , in kilobytes
<i>li</i>	The <i>line size</i> of the data cache at level <i>i</i> , in bytes
<i>ai</i>	The <i>associativity</i> of the data cache at level <i>i</i>

For example, *i*=1 designates level 1 cache properties, *s1/l1/a1*.

Defaults

If `-xcache` is not specified, the default `-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.

Examples

`-xcache=16/32/4:1024/32/1` specifies the following:

Level 1 cache has:	Level 2 cache has:
16 Kbytes	1024 Kbytes
32 bytes line size	32 bytes line size
4-way associativity	Direct mapping associativity

See also

`-xtarget=t`

`-xcg(89 | 92)`

SPARC: Compiles for generic SPARC or SPARC V8 architecture.

This option specifies the code generator for floating-point hardware in SPARC devices released in 1989 or 1992. Use the `fpversion` command to determine which floating-point hardware you have. If you compile one procedure of a program with this option, it does not mean that you must compile *all* the procedures of that program in the same way.

`-xg89` generates code to run on generic SPARC architecture.

`-xg92` generates code to run on SPARC V8 architecture.

Expansions

The option `-xcg89` is a macro for `-xtarget=ss2`, which is equivalent to `-xarch=v7 -xchip=old -xcache=64/32/1`.

The option `-xcg92` is a macro for `-xtarget=ss1000`, which is equivalent to

`-xarch=v8 -xchip=super -xcache=16/64/4:1024/64/1`.

Warnings

If you compile and link in separate steps and you compile with `-xcg89` or `-xcg92`, be sure to link with the same option, or you might get unexpected results.

`-xchip=c`

Specifies target processor for use by the optimizer.

The `-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.

Values

The following table lists the valid values for `c`.

Platform	Value of <code>c</code>	Optimize for
SPARC	<code>generic</code>	Using timing properties for good performance on most SPARC processors
SPARC	<code>old</code>	Using timing properties of processors earlier than the SuperSPARC™ chip
SPARC	<code>super</code>	Using timing properties of the SuperSPARC chip
SPARC	<code>super2</code>	Using timing properties of the SuperSPARC II chip
SPARC	<code>micro</code>	Using timing properties of the microSPARC chip
SPARC	<code>micro2</code>	Using timing properties of the microSPARC II chip

Platform	Value of <i>c</i>	Optimize for
SPARC	<code>hyper</code>	Using timing properties of the hyperSPARC™ chip
SPARC	<code>hyper2</code>	Using timing properties of the hyperSPARC II chip
SPARC	<code>powerup</code>	Using timing properties of the Weitek PowerUp chip
SPARC	<code>ultra</code>	Using timing properties of the UltraSPARC I chip
SPARC	<code>ultra2</code>	Using timing properties of the UltraSPARC II chip
SPARC	<code>ultra2i</code>	Using timing properties of the UltraSPARC Iii chip
x86	<code>generic</code>	Using timing properties of most x86 processors
x86	<code>386</code>	Using timing properties of the Intel 386 chip
x86	<code>486</code>	Using timing properties of the Intel 486 chip
x86	<code>pentium</code>	Using timing properties of the Intel Pentium chip
x86	<code>pentium_pro</code>	Using timing properties of the Intel Pentium Pro chip

Defaults

On most SPARC 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 them.

`-xcode=a`

SPARC: Specifies the code address space.

Values

a must be one of the following:

Value of <i>a</i>	Meaning
abs32	Generates 32-bit absolute addresses, which are fast, but have limited range. Code + data + bss size is limited to 2**32 bytes.
abs44	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: -xarch=(v9 v9a)
abs64	SPARC: Generates 64-bit absolute addresses, which are slow, but have full range. Available only on 64-bit architectures: -xarch=(v9 v9a)
pic13	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.
pic32	Generates position-independent code (large model), which is slow, 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.

Defaults

For SPARC V8 and V7 processors, the default is `-xcode=abs32`.

For SPARC and UltraSPARC processors (with `-xarch=(v9|v9a)`), the default is `-xcode=abs64`.

Interactions

When building shared dynamic libraries with `-xarch=v9` or `-xarch=v9a` using 64-bit Solaris 7 software, the `-xcode=pic13` or `-xcode=pic32` (or `-pic` or `-PIC`) option must be specified.

`-xF`

Enables reordering of functions by the linker.

If you compile with the `-xF` option and then run the Analyzer, you can generate a map file that shows an optimized order for the functions. A subsequent link to build

the executable file can be directed to use that map by using the linker `-Mmapfile` option. It places each function from the executable file into a separate section.

Reordering the subprograms in memory is useful only when the application text page fault time is consuming a large percentage of the application time. Otherwise, reordering might not improve the overall performance of the application.

Interactions

The `-xF` option is only supported with `-features=no%except (-noex)`.

See also

`analyzer(1)`, `debugger(1)`, `ld(1)` man pages

`-xhelp=flags`

Same as `-help`. Displays list of compiler options.

`-xhelp=readme`

Same as `-readme`. Displays contents of online readme file.

`-xildoff`

Turns off the incremental linker.

Defaults

This option is assumed if you do *not* use the `-g` option. It is also assumed if you *do* use the `-G` option, or name any source file on the command line. Override this default by using the `-xildon` option.

See also

`-xildon`, `ild(1)` man page, `ld(1)` man page, *Incremental Link Editor Guide*

`-xildon`

Turns on the incremental linker.

This option is assumed if you use `-g` and *not* `-G`, and you do not name any source file on the command line. Override this default by using the `-xildoff` option.

See also

`-xildoff`, `ild(1)` man page, `ld(1)` man page, *Incremental Link Editor Guide*

`-xinline=f[,...f]`

Inlines specified routines.

Note - This option does not affect C++ inline functions and is not related to the `+d` option.

This option instructs the optimizer to inline the user-written functions and subroutines specified in a comma-separated list. Inlining is an optimization technique whereby the compiler effectively replaces a function call with the actual subprogram code itself. Inlining often provides the optimizer more opportunities to produce efficient code.

The following restrictions apply—no warning is issued:

- The compiler determines if actual inlining is advantageous or safe.
- The source for the routine must be in the file being compiled.

Examples

```
demo% cat example.cc
static int twice ( int i ) { return 2*i; }
int main() {return twice( 3 ); }
demo% CC -compat=4 -O example.cc
demo% nm -C a.out | grep twice[37] | 68068| 8|FUNC |LOCL |0 |7 |twice(int)      [__OFFtwice]
demo% CC -compat=4 -O -xinline=__OFFtwice example.cc
```

Interactions

If a function specified in the list is not declared as `extern "C"`, the function name should be mangled. You can use the `nm` command on the executable file to find the

mangled function names. For functions declared as `extern "C"`, the names are not mangled by the compiler.

The `-xinline` option has no effect for optimization levels below `-xO3`. At `-xO4`, the optimizer decides which functions should be inlined, and does so without the `-xinline` option being specified. At `-xO4`, the compiler also attempts to determine which functions will improve performance if they are inlined. If you force inlining of a function with `-xinline`, you might actually diminish performance.

See also

`nm(1)`, `+d`

`-xlibmieee`

Causes `libm` to return IEEE 754 values for math routines in exceptional cases.

The default behavior of `libm` is XPG-compliant.

See also

Numerical Computation Guide

`-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.

See also

Numerical Computation Guide

`-xlibmopt`

Uses library of optimized math routines.

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.

Interactions

This option is implied by the `-fast` option.

See also

`-fast`, `-xnolibmopt`

`-xlic_lib=l[,...l]`

Links with the Sun-supplied, licensed library specified by *l*.

This option, like `-l`, should appear at the end of the command line, after source or object files.

Examples

To link with the Sun™ Performance Library™ :

```
demo% CC program.cc -xlic_lib=sunperf
```

See also

`performance_library` README

`-xlicinfo`

Shows license server information.

This option returns the license-server name and the user ID for each user who has a license checked out. When you use this option, the compiler is not invoked, and a license is not checked out.

If a conflicting option is used, the latest one on the command line takes precedence, and a warning is issued.

Examples

Do not compile; report license information:

```
CC -c -xlicinfo any.cc
```

Compile; do not report license information:

```
CC -xlicinfo -c any.cc
```

-Xm

Use `-features=iddollar`.

-xM

Outputs makefile dependency information.

Examples

When you compile the following code, `hello.c`,

```
#include <stdio>
using std::printf;
int main () {
    printf ("Hello World!\n");
}
```

with this option:

```
demo% CC -xM hello.c
```

the output shows the dependencies:

```
hello.o : hello.c
hello.o : /opt/SUNWsp/SC5.0/include/CC/cstdio
hello.o : /usr/include/sys/va_list.h
```

See also

make(1) (for details about makefiles and dependencies)

`-xM1`

Generates dependency information, but excludes `/usr/include`.

This is the same as `-xM`, except that this option does not report dependencies for the `/usr/include` header files.

Examples

When you compile the code sample shown for the `-xM` option:

```
% CC -xM1 hello.c
```

the output is:

```
hello.o: hello.c
```

`-xMerge`

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`.

See also

`ld(1)` man page

`-xnoLib`

Disables linking with default system libraries.

Normally (without this option) the C++ compiler links with several system libraries to support C++ programs. With this option, the `-llib` options to link the default system support libraries are *not* passed to `ld`.

Normally, the compiler links with the following libraries in the following order:

For `compat=4`:

```
-lC -lC_mtstubs -lm -lw -lcx -lc
```

For `compat=5`:

```
-lCstd -lCrun -lC_mtstubs -lm -lw -lcx -lc
```

The order of the `-l` options is significant. The `-lm`, `-lw`, and `-lcx` options must appear before `-lc`.

Note - If the `-mt` compiler option is specified, then `-lthread` is passed instead of `-lC_mtstubs`.

Examples

For minimal compilation to meet the C application binary interface (that is, a C++ program with only C support required) use:

```
demo% CC -xnolib test.cc -lc
```

To link `libm` statically into a single-threaded application on SPARC V8 architecture, use:

```
demo% CC -xnolib -Bstatic -lm -Bdynamic -lC_mtstubs -lw -lcx -lc
```

To link `libm` and `libw` statically, and link others dynamically:

```
demo% CC -xnolib -Bstatic -lm -lw -Bdynamic -lcx -lc
```

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

This set of libraries is not stable and might change from release to release.

`-lcx` is not present in 64-bit compilation modes.

See also

`-library`, `-staticlib`, `-l`

`-xnolibmil`

Cancels `-xlibmil` on the command line.

Use this option with `-fast` to override linking with the optimized math library.

`-xno libmopt`

Does not use the math routine library.

Examples

Use this option after the `-fast` option on the command line, as in:

```
demo% CC -fast -xno libmopt
```

`-xO [level]`

Specifies optimization level. In general, program execution speed depends on level of optimization. The higher the level of optimization, the faster the speed.

If `-xO [level]` is not specified, only a very basic level of optimization (limited to local common subexpression elimination and dead code analysis) is performed. A program's performance might be significantly improved when it is compiled with an optimization level. Use of `-O` (which implies `-xO2`) is recommended for most programs.

Generally, the higher the level of optimization with which a program is compiled, 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` 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.

Values

On the SPARC Platform:

`-xO` is equivalent to `-xO2`.

`-xO1` does only the minimum amount of optimization (peephole), which is postpass, 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
- 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. In general, this level results in minimum code size.

`-x03`, in addition to optimizations performed at the `-x02` 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 `-x02`. In general, `-x03` results in increased code size. If you are running out of swap space, use `-x02`.

`-x04` does automatic inlining of functions contained in the same file in addition to performing `-x03` optimizations. This automatic inlining usually improves execution speed but sometimes makes it worse. In general, this level results in increased code size.

`-x05` 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 “`-xprofile=p`” on page 99.

On the x86 Platform:

`-x01` preloads arguments from memory and causes cross jumping (tail merging), as well as the single pass of the default optimization.

`-x02` schedules both high- and low-level instructions and performs improved spill analysis, loop memory-reference elimination, register lifetime analysis, enhanced register allocation, global common subexpression elimination, as well as the optimization done by level 1.

`-x03` performs loop strength reduction and inlining, as well as the optimization done by level 2.

`-xO4` performs architecture-specific optimization, as well as the optimization done by level 3.

`-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

Debugging with `-g` does not suppress `-xO[level]`, but `-xO[level]` limits `-g` in certain ways.

The `-xO3` and `-xO4` 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.

The `-xinline` option has no effect for optimization levels below `-xO3`. At `-xO4`, the optimizer decides which functions should be inlined, and does so without the `-xinline` option being specified. At `-xO4`, the compiler also attempts to determine which functions will improve performance if they are inlined. If you force inlining of a function with `-xinline`, you might actually diminish performance.

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 `cs(1)` man page). For example, to limit virtual memory to 16 megabytes:

```
demo% limit datasize 16M
```

This command causes the optimizer to try to recover if it reaches 16 megabytes 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

`-fast`, `-xprofile=p`, *cs(1)* man page

`-xpg`

Compiles for profiling with the `gprof` profiler.

The `-xpg` option compiles self-profiling code to collect data for profiling with `gprof`. This option invokes a runtime recording mechanism that produces a `gmon.out` file when the program normally terminates.

Warnings

If you compile and link separately, and you compile with `-xpg`, be sure to link with `-xpg`.

See also

`-xprofile=p`, *analyzer(1)* man page

`-xprefetch [= (yes | no)]`

SPARC: Uses prefetch instructions on UltraSPARC II processors.

With `-xprefetch=yes`, the compiler is free to insert prefetch instructions into the code it generates. This can result in a performance improvement on UltraSPARC II processors.

Defaults

If the `-xprefetch` option is not specified, the default is `-xprefetch=no`.

Specifying `-xprefetch` alone is equivalent to `-xprefetch=yes`.

`-xprofile=p`

Collects or optimizes with runtime profiling data.

This option causes execution frequency data to be collected and saved during the execution. The data can then be used in subsequent runs to improve performance. This option is valid only when a level of optimization is specified.

Values

p must be one of the following:

Value of <i>p</i>	Meaning
<code>collect[:name]</code>	<p>Collects and saves execution frequency for later use by the optimizer with <code>-xprofile=use</code>. The compiler generates code to measure statement execution frequency. The <i>name</i> is the name of the program that is being analyzed. This <i>name</i> is optional. If not specified, <code>a.out</code> is assumed to be the name of the executable.</p> <p>At runtime, a program compiled with <code>-xprofile=collect:name</code> creates the subdirectory <code>name.profile</code> to hold the runtime feedback information. Data is written to the file <code>feedback</code> in this subdirectory. If you run the program several times, the execution frequency data accumulates in the <code>feedback</code> file; that is, output from prior runs is not lost.</p>
<code>use[:name]</code>	<p>Uses execution frequency data to optimize strategically. The <i>name</i> is the name of the executable that is being analyzed. The <i>name</i> is optional and, if not specified, is assumed to be <code>a.out</code>.</p> <p>The program is optimized by using the execution frequency data previously generated and saved in feedback files that were written by a previous execution of the program compiled with <code>-xprofile=collect</code>.</p> <p>The source files and other compiler options must be exactly the same as those used for the compilation that created the compiled program that generated the feedback file. If compiled with <code>-xprofile=collect:name</code>, the same program name, <i>name</i>, must appear in the optimizing compilation: <code>-xprofile=use:name</code>.</p>
<code>tcov</code>	<p>Basic block coverage analysis using the new style <code>tcov</code>.</p> <p>This option is the new style of basic block profiling for <code>tcov</code>. It has similar functionality to the <code>-xa</code> 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 <code>-xa</code> option, but <code>.d</code> 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 <code>/foo/bar/myprog.profile</code>, then the data file is stored in <code>/foo/bar/myprog.profile/myprog.tcovd</code>.</p> <p>When running <code>tcov</code>, you must pass it the <code>-x</code> option to force it to use the new style of data. If you don't, <code>tcov</code> uses the old <code>.d</code> files by default, and produces unexpected output.</p> <p>Unlike the <code>-xa</code> option, the <code>TCOVDIR</code> environment variable has no effect at compile time. However, its value is used at program runtime.</p>

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.

`-xprofile=tcov` is incompatible with `-g`.

Warnings

If compilation and linking are performed in separate steps, the same `-xprofile` option must appear in the compile as well as the link step.

See also

`-xa`, `tcov(1)` *man page*, *Analyzing Program Performance With Sun WorkShop*

`-xregs=r[,...r]`

SPARC: 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.

Values

`r` must be one of the following (meaning depends upon the `-xarch` setting):

Value of <i>r</i>	Meaning
[no%]appl	<p>[Does not] Allows use of registers g2, g3, and g4 (v8, v8a)</p> <p>[Does not] Allows use of registers g2, g3, g4, and g5 (v8plus, v8plusa)</p> <p>[Does not] Allows use of registers g2, g3 (v9, v9a)</p> <p>In the SPARC ABI, these registers are described as <i>application</i> registers. Using these registers can increase performance because fewer <code>load</code> and <code>store</code> instructions are needed. However, such use can conflict with programs that use the registers for other purposes.</p>
[no%]float	<p>[Does not] Allows use of floating-point registers as specified in the SPARC ABI.</p> <p>You can use the floating-point registers even if the program contains no floating point code.</p> <p>With the <code>no%float</code> option a source program cannot contain any floating-point code.</p>

Defaults

If `-xregs` is not specified, `-xregs=appl,float` is assumed.

Examples

To compile an application program using all available scratch registers, use:

```
-xregs=appl,float
```

To compile non-floating-point code sensitive to context switch, use:

```
-xregs=no%appl,no%float
```

See also

SPARC V7/V8 ABI, SPARC V9 ABI

-XS

Allows debugging by `dbx` without object (`.o`) files.

This option disables Auto-Read for `dbx`. Use this option if you cannot keep the `.o` files around. This option passes the `-s` option to the assembler.

No Auto-Read is the older way of loading symbol tables. It places all symbol tables for `dbx` in the executable file. The linker links more slowly, and `dbx` initializes more slowly.

Auto-Read is the newer and default way of loading symbol tables. With Auto-Read the information is placed in the `.o` files, so that `dbx` loads the symbol table information only if it is needed. Hence the linker links faster, and `dbx` initializes faster.

With `-xs`, if you move executables to another directory, you do not have to move the object (`.o`) files to use `dbx`.

Without `-xs`, if you move the executables to another directory, you must move *both* the source files and the object (`.o`) files to use `dbx`.

`-xsafe=mem`

SPARC: Allows no memory-based traps to occur.

This option grants permission to use the speculative load instruction on V9 machines.

Interactions

This option is only effective if used with `-xO5` optimization when `-xarch=v8plus`, `v8plusa`, `v9`, or `v9a` is specified.

Warnings

You should use this option only if you can safely assert that no memory-based traps occur in your program. For most programs, this assertion is appropriate and can be safely made. For a program that explicitly forces memory-based traps to handle exceptional conditions, this assertion is not safe.

`-xsb`

Produces information for the Sun WorkShop source browser.

This option causes the `CC` driver to generate extra symbol table information in the `SunWS_cache` subdirectory for the source browser.

See also

-xsbfast

-xsbfast

Produces *only* source browser information, no compilation.

This option runs only the `ccfe` phase to generate extra symbol table information in the `SunWS_cache` subdirectory for the source browser. No object file is generated.

See also

-xsb

-xspace

SPARC: Does not allow optimizations that increase code size.

-xtarget=*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.

Values (SPARC)

t must be one of the following:

Value of <i>t</i>	Meaning
<code>native</code>	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.
<code>generic</code>	Gets the best performance for generic architecture, chip, and cache. The compiler expands <code>-xtarget=generic</code> to: <code>-xarch=generic -xchip=generic -xcache=generic</code> This is the default value.
<i>platform-name</i>	Gets the best performance for the specified platform. Select a SPARC platform name from the table in the following section.

Values (x86)

For the x86 platform, `-xtarget` accepts the following values:

- `native` or `generic`
- `386`—Directs the compiler to generate code for the best performance on the Intel 80386 microprocessor.
- `486`—Directs the compiler to generate code for the best performance on the Intel 80486 microprocessor.
- `pentium`—Directs the compiler to generate code for the best performance on the Pentium or Pentium Pro microprocessor.
- `pentium_pro`—Directs the compiler to generate code for the best performance on the Pentium Pro microprocessor.

<code>-xtarget</code>	<code>-xarch</code>	<code>-xchip</code>	<code>-xcache</code>
<code>cs6400</code>	<code>v8</code>	<code>super</code>	<code>16/32/4:2048/64/1</code>
<code>entr2</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:512/64/1</code>
<code>entr2/1170</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:512/64/1</code>
<code>entr2/1200</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:512/64/1</code>

-xtarget	-xarch	-xchip	-xcache
entr2/2170	v8	ultra	16/32/1:512/64/1
entr2/2200	v8	ultra	16/32/1:512/64/1
entr150	v8	ultra	16/32/1:512/64/1
entr3000	v8	ultra	16/32/1:512/64/1
entr4000	v8	ultra	16/32/1:512/64/1
entr5000	v8	ultra	16/32/1:512/64/1
entr6000	v8	ultra	16/32/1:512/64/1
sc2000	v8	super	16/32/4:2048/64/1
solb5	v7	old	128/32/1
solb6	v8	super	16/32/4:1024/32/1
ss1	v7	old	64/16/1
ss1plus	v7	old	64/16/1
ss2	v7	old	64/32/1
ss2p	v7	powerup	64/32/1
ss4	v8a	micro2	8/16/1
ss4/85	v8a	micro2	8/16/1
ss4/110	v8a	micro2	8/16/1
ss5	v8a	micro2	8/16/1
ss5/85	v8a	micro2	8/16/1

-xtarget	-xarch	-xchip	-xcache
ss5/110	v8a	micro2	8/16/1
ss10	v8	super	16/32/4
ss10/20	v8	super	16/32/4
ss10/30	v8	super	16/32/4
ss10/40	v8	super	16/32/4
ss10/41	v8	super	16/32/4:1024/32/1
ss10/50	v8	super	16/32/4
ss10/51	v8	super	16/32/4:1024/32/1
ss10/61	v8	super	16/32/4:1024/32/1
ss10/71	v8	super2	16/32/4:1024/32/1
ss10/402	v8	super	16/32/4
ss10/412	v8	super	16/32/4:1024/32/1
ss10/512	v8	super	16/32/4:1024/32/1
ss10/514	v8	super	16/32/4:1024/32/1
ss10/612	v8	super	16/32/4:1024/32/1
ss10/712	v8	super2	16/32/4:1024/32/1
ss10/hs11	v8	hyper	256/64/1
ss10/hs12	v8	hyper	256/64/1
ss10/hs14	v8	hyper	256/64/1

-xtarget	-xarch	-xchip	-xcache
ss10/hs21	v8	hyper	256/64/1
ss10/hs22	v8	hyper	256/64/1
ss20	v8	super	16/32/4:1024/32/1
ss20/50	v8	super	16/32/4
ss20/51	v8	super	16/32/4:1024/32/1
ss20/61	v8	super	16/32/4:1024/32/1
ss20/71	v8	super2	16/32/4:1024/32/1
ss20/151	v8	hyper	512/64/1
ss20/152	v8	hyper	512/64/1
ss20/502	v8	super	16/32/4
ss20/514	v8	super	16/32/4:1024/32/1
ss20/612	v8	super	16/32/4:1024/32/1
ss20/712	v8	super2	16/32/4:1024/32/1
ss20/hs11	v8	hyper	256/64/1
ss20/hs12	v8	hyper	256/64/1
ss20/hs14	v8	hyper	256/64/1
ss20/hs21	v8	hyper	256/64/1
ss20/hs22	v8	hyper	256/64/1
ss600/41	v8	super	16/32/4:1024/32/1

-xtarget	-xarch	-xchip	-xcache
ss600/51	v8	super	16/32/4:1024/32/1
ss600/61	v8	super	16/32/4:1024/32/1
ss600/120	v7	old	64/32/1
ss600/140	v7	old	64/32/1
ss600/412	v8	super	16/32/4:1024/32/1
ss600/512	v8	super	16/32/4:1024/32/1
ss600/514	v8	super	16/32/4:1024/32/1
ss600/612	v8	super	16/32/4:1024/32/1
ss1000	v8	super	16/32/4:1024/32/1
sselc	v7	old	64/32/1
ssipc	v7	old	64/16/1
ssipx	v7	old	64/32/1
sslc	v8a	micro	2/16/1
sslt	v7	old	64/32/1
sslx	v8a	micro	2/16/1
sslx2	v8a	micro2	8/16/1
ssslc	v7	old	64/16/1
ssvygr	v8a	micro2	8/16/1
sun4/15	v8a	micro	2/16/1

-xtarget	-xarch	-xchip	-xcache
sun4/20	v7	old	64/16/1
sun4/25	v7	old	64/32/1
sun4/30	v8a	micro	2/16/1
sun4/40	v7	old	64/16/1
sun4/50	v7	old	64/32/1
sun4/60	v7	old	64/16/1
sun4/65	v7	old	64/16/1
sun4/75	v7	old	64/32/1
sun4/110	v7	old	2/16/1
sun4/150	v7	old	2/16/1
sun4/260	v7	old	128/16/1
sun4/280	v7	old	128/16/1
sun4/330	v7	old	128/16/1
sun4/370	v7	old	128/16/1
sun4/390	v7	old	128/16/1
sun4/470	v7	old	128/32/1
sun4/490	v7	old	128/32/1
sun4/630	v7	old	64/32/1
sun4/670	v7	old	64/32/1

<code>-xtarget</code>	<code>-xarch</code>	<code>-xchip</code>	<code>-xcache</code>
<code>sun4/690</code>	<code>v7</code>	<code>old</code>	<code>64/32/1</code>
<code>ultra</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:512/64/1</code>
<code>ultra1/140</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:512/64/1</code>
<code>ultra1/170</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:512/64/1</code>
<code>ultra1/200</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:512/64/1</code>
<code>ultra2</code>	<code>v8</code>	<code>ultra2</code>	<code>16/32/1:512/64/1</code>
<code>ultra2/1170</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:512/64/1</code>
<code>ultra2/1200</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:1024/64/1</code>
<code>ultra2/1300</code>	<code>v8</code>	<code>ultra2</code>	<code>16/32/1:2048/64/1</code>
<code>ultra1/2170</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:512/64/1</code>
<code>ultra1/2200</code>	<code>v8</code>	<code>ultra</code>	<code>16/32/1:1024/64/1</code>
<code>ultra1/2300</code>	<code>v8</code>	<code>ultra2</code>	<code>16/32/1:2048/64/1</code>
<code>ultra2i</code>	<code>v8</code>	<code>ultra2i</code>	<code>16/32/1:512/64/1</code>

For SPARC devices, `-xtarget` accepts the following expansion values:

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

Compiling for 64-bit Solaris 7 on SPARC or UltraSPARC V9 devices is indicated by the `-xarch=v9|v9a` option. Setting `-xtarget=ultra` or `ultra2` is not necessary or sufficient. If `-xtarget` is specified, the `-xarch=v9` or `v9a` option must appear after the `-xtarget`, as in:

```
-xtarget=ultra -xarch=v9
```

Otherwise, the `-xtarget` setting reverts the `-xarch` value to `v8`.

`-xtime`

Causes the `CC` driver to report execution time for the various compilation passes.

`-xunroll=n`

Enables unrolling of loops where possible.

This option specifies whether or not the compiler optimizes (unrolls) loops.

Values

When *n* is 1, it is a suggestion to the compiler to not unroll loops.

When *n* is an integer greater than 1, `-unroll=n` causes the compiler to unroll loops *n* times.

`-xwe`

Converts all warnings to errors by returning nonzero exit status.

`-ztext`

Forces a fatal error if any relocations remain against nonwritable, allocatable sections.

This option is passed to the linker.

Compiling Templates

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.

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.

Template Commands

The `CCadmin(1)` command administers the template repository. For example, changes in your program can render some instantiations superfluous, thus wasting storage space. The `CCadmin -clean` command (formerly `ptclean`) clears out all instantiations and associated data. Instantiations are recreated only when needed.

Template Instance Placement and Linkage

You can instruct the compiler to use one of five instance placement and linkage methods: external, static, global, explicit, and semi-explicit.

- External instances are suitable for all development and provide the best overall template compilation.
- Static instances are suitable for very small programs or debugging and have restricted uses.
- Global instances are suitable for some library construction.
- 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.

You should use the external instances method, which is the default, unless there is a very good reason to do otherwise. See the *C++ Programming Guide* for further information.

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.

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

Specify external linkage with the `-instances=extern` option (the default 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:

```
%demo CC -xar -o libmain.a a.o b.o c.o
```

See Chapter 6 for more information.

Static Instances

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.

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. This has the following undesirable consequences:

- Multiple instances produce unnecessarily large programs. (Static instance linkage is therefore suitable only for small programs, where templates are unlikely to be multiply instantiated.)
- Templates that contain static variables have many copies of the variable, and this is an unavoidable violation of the C++ standard. Therefore, use of static instances is not supported with static variables within templates.

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*.)

Specify static instance linkage with the `-instances=static` compiler option. You can use static instance linkage only with the definitions-included template organization. The compiler does not search for definitions. (See *C++ Programming Guide*.)

Global Instances

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

Template instances receive global linkage. These instances are visible and usable outside the current compilation unit. As a consequence, instantiation in more than one compilation unit results in multiple symbol definition errors during linking. The global instances method is therefore suitable only when you know that instances will not be repeated.

Specify global instances with the `-instances=global` option.

Global instances can be used only with the definitions-included template organization. The compiler does not search for definitions.

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. As a consequence, templates are reinstantiated during each recompilation; they are not saved to the template repository.

Template instances receive global linkage. These instances are visible and usable outside the current compilation unit. Multiple explicit instantiations within a program result in multiple symbol definition errors during linking. The explicit instances method is therefore suitable only when you know that instances are not repeated, such as when you construct libraries with explicit instantiation.

Specify explicit instances with the `-instances=explicit` option.

You can use explicit instance linkage only with the definitions-included template organization. The compiler does not search for definitions.

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. 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; they are not saved to the template repository.

Explicit instances receive global linkage. These instances are visible and usable outside the current compilation unit. Multiple explicit instantiations within a program result in multiple symbol definition errors during linking. The semi-explicit instances method is therefore suitable only when you know that explicit instances will not be repeated, such as when you construct libraries with explicit instantiation.

Implicit instances used from within the bodies of explicit instances receive static linkage. These instances are not visible outside the current compilation unit. As a result, templates can have identical instantiations in several object files. This has two undesirable consequences.

- Multiple instances produce unnecessarily large programs. (Semi-explicit instance linkage is therefore suitable only for programs where template bodies do not cause multiple instantiations.)
- Templates that contain static variables have many copies of the variable; this is an unavoidable violation of the C++ standard. Therefore, use of the semi-explicit instances method is not supported with static variables within templates.

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

You can use semi-explicit instance linkage only with the definitions-included template organization. The compiler does not search for definitions.

The Template Repository

The template repository stores template instances between separate compilations so that template instances are compiled only when 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.

Repository Structure

The template repository is contained, by default, within the Sun WorkShop Cache directory (`SunWS_cache`). The Sun WorkShop Cache directory is contained within the directory in which the output files will be placed. You can change the name of the cache directory by setting the `SUNWS_CACHE_NAME` environment variable.

Writing to the Template Repository

When the compiler must store template instances, it stores them within the template repository corresponding to the output file. That is,

```
demo% CC -o sub/a.o a.cc
```

this 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 directory is created for you.

You can specify an alternate repository location with the `-ptrdirectory` option.

Reading From Multiple Template Repositories

The compiler reads from the template repositories corresponding to the object files that it reads. That is,

```
demo% CC sub1/a.o sub2/b.o
```

this command line:

reads from `./sub1/SunWS_cache` and `./sub2/SunWS_cache`, and, if necessary, writes to `./SunWS_cache`.

Do not specify multiple repositories by giving multiple `-ptr` options. You can only use `-ptr` to specify a single repository location.

Sharing Template Repositories

Do not share template repositories among multiple programs or libraries. That is, the following approach is not supported and can result in inconsistent results and random link errors.

```
demo% cc a.cc  
demo% cc b.cc
```

In practice, this means you must either compile separate programs (or libraries) in separate directories, or clean out the template repository between each program or library compilation.

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 the *C++ Programming Guide*.

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 base name 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`, or `.cxx`). The template definition file must be located in one of the normal `include` directories or in the same directory as its matching header file.

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.

Template Instance Automatic Consistency

The template repository manager ensures that the states of the instances in the repository are consistent and up-to-date with your source files.

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.

Compile-Time Instantiation

Instantiation is the process by which a C++ compiler creates a usable function or object from a template. The Sun C++ 5.0 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.

Using Libraries

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

The C Libraries

The Solaris operating environment comes with several libraries installed in `/usr/lib`. Most of these libraries have a C interface. Of these, the `libc`, `libm`, and `libw` 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:

```
demo% CC text.c -ldemangle
```

The Sun C++ 5.0 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. See “Libraries Provided With the C++ Compiler” on page 124 for more information.

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 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 shipped with the C++ compiler and the modes in which they are available.

Library	Description	Available Mode (s)
<code>libCrun</code>	C++ runtime	<code>-compat=5</code>
<code>libCstd</code>	C++ Standard Library	<code>-compat=5</code>
<code>libiostream</code>	Classic iostreams	<code>-compat=5</code>
<code>libc</code>	C++ runtime, classic iostreams	<code>-compat=4</code>
<code>libC_mtstubs</code>	mtstubs library	<code>-compat=4</code> , <code>-compat=5</code>
<code>libcomplex</code>	complex library	<code>-compat=4</code>
<code>librwtool</code>	<code>Tools.h++ 7</code>	<code>-compat=4</code> , <code>-compat=5</code>
<code>librwtool_dbg</code>	Debug-enabled <code>Tools.h++ 7</code>	<code>-compat=5</code>
<code>libgc</code>	Garbage collection	C interface
<code>libgc_dbg</code>	Debug-enabled garbage collection	<code>-compat=4</code> , <code>-compat=5</code>
<code>libdemangle</code>	Demangling	C interface

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* manual for details.

`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.

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

`libC_mtstubs`: This library contains stubs of some `libthread` functions. It gets linked in for single threaded applications. If you build your application with the `-mt` option, `libthread` gets linked in place of `libC_mtstubs`.

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

`librwtool`: (Tools.h++ 7) This is Rogue Wave's Tools.h++ version 7 library.

`libgc`: This is the garbage collection library (a component of Sun WorkShop Memory Monitor). For further details about this library, point your web browser to the file:

```
/opt/SUNWspro/SC5.0/htmldocs/locale/C/gc/start.html
```

If you installed the compiler in a different directory, point your browser to:

```
install-directory/SC5.0/htmldocs/locale/C/gc/start.html
```

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

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: `-lCstd -lCrun -lC_mtstubs -lm -lw -lcx -lc`

In compatibility mode, the following libraries are linked by default: `-lC -lC_mtstubs -lm -lw -lcx -lc`

See “`-library=I[,...I]`” on page 63 for more information.

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 `-library` option to specify the following libraries that are shipped with the Sun C++ compiler:
 - `libCrun`
 - `libCstd`
 - `libiostream`
 - `libC`
 - `libcomplex`
 - `librwtool, librwtool_dbg`
 - `libgc, libgc_dbg`

A library that is specified with both `--library` and `--staticlib` options will be linked statically. Some examples:

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

links the `Tools.h++` version 7 and `libiostream` libraries dynamically.

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

links the `libgc` library statically.

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

compiles `test.cc` in compatibility mode and links `libC` statically. Because `libC` is linked by default in compatibility mode, you do not have to specify this library with the `-library` option.

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

excludes the libraries `libCrun` and `libCstd`, which would otherwise be included by default.

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/opt/SUNWspro/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.

Using Class Libraries

Generally, two steps are involved in using a class library. First, include the appropriate header in your source code. Second, link your program with the object library.

The `iostream` Library

The C++ 5.0 compiler provides two implementations of `iostreams`:

- **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 all 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`.

```
// 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.

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

The next example uses standard `iostreams`.

```
// file prog1.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.

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

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. With 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 would 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.

```
// 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.

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

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

```
// file ex2.cc (ex1.cc rewritten for standard mode)
#include <iostream>
#include <complex>
int main()
{
    std::complex<double> x(3,3), y(4,4);
    std::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.

```
% CC ex2.cc
% a.out
x=(3,3), y=(4,4), z=(0,24)
```

Linking C++ Libraries

The following table shows the compiler options for linking the C++ libraries. See “`-library=I[,...I]`” on page 63 for more information.

Library	Compile Mode	Option
Classic <code>iostream</code>	-compat=4	None needed
	-compat=5	-library=iostream
complex	-compat=4	-library=complex
	-compat=5	None needed
Tools.h++ v7	-compat=4	-library=rwtool7
	-compat=5	-library=rwtool7,iostream
Tools.h++ v7 debug	-compat=4	-library=rwtool7_dbg
	-compat=5	- library=rwtool7_dbg,iostream

Library	Compile Mode	Option
Garbage collection	-compat=4	-library=gc
	-compat=5	-library=gc
Garbage collection debug	-compat=4	-library=gc_dbg
	-compat=5	-library=gc_dbg

Statically Linking Standard Libraries

The `CC` driver links in shared versions of several libraries by default, including `libc` and `libm`, by passing a `-llib` option for each of the default libraries to the linker. (See “Default C++ Libraries” on page 125 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 `-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 `libw`, `libm`, and `libc` in the Solaris 2.5.1, 2.6, or Solaris 7 environment:

```
demo% CC test.c -xnolib -lCstd -Bstatic -lCrun \
-Bdynamic -lC_mtstubs -lm -lw -lcx -lc
```

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

Note - The `-lcx` option does not exist on the x86 platform.

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 `-lthread` to `ld` in place of `-lC_mtstubs`. However, if you use both `-mt` and `-xnolib`, the `CC` driver does not pass `-lthread` to `ld`. See “`-xnolib`” on page 94 for more information. See *Linker and Libraries Guide* for more information about `ld`.

You can also use the `-library` option along with the `-staticlib` option to link a C++ library statically. This alternative is much easier than the one described earlier. The previous example, for instance, can be performed as:

```
demo% CC test.c -staticlib=Crun
```

Using Shared Libraries

The following shared libraries are included with the C++ compiler:

- libCrun.so.1
- libC.so.5
- libcomplex.so.5
- librwtool.so.2
- libgc.so.1
- libgc_dbg.so.1

The occurrence of each shared object linked with the program is recorded in the resulting executable (a.out file); this information is used by ld.so to perform dynamic link editing at runtime. Because the work of incorporating the library code into an address space is deferred, the runtime behavior of the program using a shared library is sensitive to an environment change—that is, moving a library from one directory to another. For example, if your program is linked with libcomplex.so.5 in /opt/SUNWspr0/SC5.0/lib in the Solaris 2.6 environment, and the libcomplex.so.5 library is later moved into /opt2/SUNWspr0/SC5.0/lib, the following message is displayed when you run the binary code:

```
ld.so: libcomplex.so.5: not found
```

You can still run the old binary code without recompiling it by setting the environment variable LD_LIBRARY_PATH to the new library directory.

In a C shell:

```
demo% setenv LD_LIBRARY_PATH \  
/opt2/SUNWspr0/SC5.0/lib:${LD_LIBRARY_PATH}
```

In a Bourne shell:

```
demo$ LD_LIBRARY_PATH=/opt2/SUNWspr0/SC5.0/lib:${LD_LIBRARY_PATH}  
demo$ export LD_LIBRARY_PATH
```

The LD_LIBRARY_PATH has a list of directories, usually separated by colons. When you run a C++ program, the dynamic loader searches the directories in LD_LIBRARY_PATH before the default directories.

Use the `ldd` command as shown in the following example to see which libraries are linked dynamically in your executable:

```
% ldd a.out
```

This step should rarely be necessary, because the shared libraries are seldom moved.

Note - When shared libraries are opened with `dlopen`, `RTLD_GLOBAL` must be used for exceptions to work.

See *Linker and Libraries Guide* for more information on using shared libraries.

Standard Header Implementation

C has 17 standard headers (`<stdio.h>`, `<string.h>`, `<stdlib.h>`, and others). These headers are delivered as part of Solaris, 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`. For logistical reasons, 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>`) with the various declared names appearing only in namespace `std`. Finally, C++ adds 32 of its own standard headers (`<string>`, `<utility>`, `<iostream>`).

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 header `<string>` would refer to a file named `string` in some directory. That obvious implementation has the following drawbacks:

- You cannot search for just header files or create a makefile rule for them if they do not have file name suffixes.
- If you put `-I/usr/include` on the compiler command line, you don't get the correct version of the standard C headers because `/usr/include` is searched before the compiler's own include directory.
- If you have a directory or executable program named `string`, it might erroneously be found instead of the standard header file.
- 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 `".SUNWCCh."` (`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.

Source code	Compiler searches for	Comments
<code><string></code>	<code>string.SUNWCCh</code>	C++ string templates
<code><cstring></code>	<code>cstring.SUNWCCh</code>	A version of C <code>string.h</code>
<code><string.h></code>	<code>string.h.SUNWCCh</code>	C <code>string.h</code>
<code><fcntl.h></code>	<code>fcntl.h</code>	Not a standard C or C++ header
<code>"string"</code>	<code>string</code>	Quotes, not angle bracket
<code><../string></code>	<code>../string</code>	Path specified

Building Libraries

This chapter explains how to build your own libraries.

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 generally ends with a `.so` suffix or `.so.number` suffix, which specifies a version number.

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 environment 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.)

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:

```
% 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. Use of the `-xar` flag is only for creating or updating an existing archive, not for maintaining the archive. It is equivalent to specifying `ar -cr`.

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.

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. Furthermore, the C++ compiler does not initialize global variables if they are defined in a dynamic library, unless the library is built correctly. All static constructors and destructors are called from the `.init` and `.fini` sections, respectively. All static constructors in a dynamic library linked to an application are called *before* `main()` is executed. Finally, exceptions might not work, unless you use the `CC -G` command to build the dynamic library.

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, `libfoo.so.1`, containing objects from source files `lsrc1.cc` and `lsrc2.cc`, type:

```
% CC -G -o libfoo.so.1 -h libfoo.so.1 -Kpic lsrc1.cc lsrc2.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 a name for the shared library. The `-Kpic` option specifies that the object files are to be position-independent.

Building Shared Libraries With Exceptions

When shared libraries are opened with `dlopen()`, you must use `RTLD_GLOBAL` for exceptions to work.

Note - When building shared libraries with exceptions in them, do not pass the option `-Bsymbolic` to `ld`. Exceptions that should be caught might be missed.

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.

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.

Building a Library With 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 you also want to remove any dependency on the C++ runtime libraries, you should enforce the following coding rules in your library sources:

- Do not use `new/delete` unless you define your own global operators `new` and `delete`.
- Do not use array `new/delete`.
- Do not use exceptions.
- Do not use RTTI.

Using `dlopen` to Access a C++ Library From a C Program

If you want to `dlopen()` 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:

```
demo% CC -G -compat=4 ... -lC
demo% CC -G -compat=5 ... -lCrun
```

If the shared library uses exceptions and does not have a dependency on the C++ runtime library, your C program might behave erratically.

Note - When shared libraries are opened with `dlopen`, `RTLD_GLOBAL` must be used for exceptions to work.

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 `-D_REENTRANT` to the preprocessor and passes `-pthread` in the correct order to `ld`. For `-compat=4`, the `-mt` option ensures that `libthread` is linked before `libc`. You should not link your application directly with `-pthread` as this causes `libthread` to be linked in an incorrect order.

The following example shows the correct way of building a multithreaded application:

```
demo% CC -c -mt myprog.cc
demo% CC -mt myprog.o
```

The following example shows the wrong way of building a multithreaded application:

```
demo% CC -c -mt myprog.o
demo% CC myprog.o -pthread
```

Indicating Multithreaded Compilation

You can check whether an application is linked to `libthread` or not by using the `ldd` command:

```
demo% CC -mt myprog.cc
demo% ldd a.out
libm.so.1 => /usr/lib/libm.so.1
libcrun.so.1 => /usr/lib/libcrun.so.1
libw.so.1 => /usr/lib/libw.so.1
libthread.so.1 => /usr/lib/libthread.so.1
libc.so.1 => /usr/lib/libc.so.1
libdl.so.1 => /usr/lib/libdl.so.1
```

Using `libc` 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
- Exceptions

Glossary

abstract class	A class that has at least one pure virtual function, and that can be used only as a base class of another class. An abstract class can have no objects of its own created; only objects of a class derived from it.
ANSI C	American National Standards Institute's definition of the C programming language. It is the same as the ISO (International Standards Organization) definition.
ANSI/ISO C++	The American National Standards Institute and the International Standards Organization standard for the C++ programming language.
array	A data structure that stores a collection of values of a single data type consecutively in memory. Each value is accessed by its position in the array.
base class	See <i>inheritance</i> .
binary compatibility	The ability to link object files compiled by one release while using a compiler of a different release.
binding	Tying a function call to a specific function definition. More generally, tying a name to a particular entity.
cfront	A C++ to C compiler program that translates C++ to C source code, which in turn can be compiled by a standard C compiler.
class	A user-defined data type consisting of named data elements (which may be of different types), and a set of operations that can be performed with the data.
class template	A template that describes a set of classes or related data types.

compiler option	An instruction to the compiler that changes its behavior. For example, the <code>-g</code> option tells the compiler to generate data for the debugger. Synonyms: <i>flag</i> , <i>switch</i> .
constructor	A special class member function 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.
data member	An element of a class that is <i>data</i> , 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 allocated to a variable and the operations that can be performed on the variable.
derived class	See <i>inheritance</i> .
destructor	A special class member function automatically called by the compiler whenever a class object is destroyed or the operator <code>delete</code> 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 <i>constructor</i> .
dynamic binding	Connection of the function call to the function body at runtime. Occurs only with virtual functions. Also called <i>late binding</i> , <i>runtime binding</i> .
dynamic cast	A safe method of converting a pointer or reference from its declared type to any type consistent with the dynamic type to which it refers.
dynamic type	The actual type of an object accessed by a pointer or reference that might have a different declared type.
ELF file	Executable and Linking Format file, 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, invoked automatically when an exception occurs for which the handler has been registered.

exception handling	An error recovery process, 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 registered at an earlier point in the execution, and the code containing the error is bypassed.
function overloading	Giving the same name, but different argument types and numbers, to different functions. Also called <i>functional polymorphism</i> .
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.
idempotent	The property of a header file that including it many times in one translation unit has an effect no different from including it once.
incremental linker	A linker that creates a new executable file by linking only the changed .o files to the previous executable.
inheritance	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.
inline function	A function that replaces the function call with the actual function code.
instantiation	The process by which a C++ compiler creates a usable function or object (instance) from a template.
ISO	International Organization for Standardization.
K&R C	The de facto C programming language standard developed by Brian Kernighan and Dennis Ritchie prior to ANSI C.
keyword	A word that has unique meaning in a programming language, and that can be used only in a specialized context.
linker	The tool that connects object code and libraries to form a complete, executable program.

lvalue	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.
mangle	See <i>name mangling</i> .
member function	An element of a class that is a function, as opposed to a data or type definition.
method	In some object-oriented languages, another name for a member function.
multiple inheritance	Inheritance of a derived class directly from more than one base class.
multithreading	The software technology that enables the development of parallel applications, whether on single- or multiple-processor systems.
name mangling	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 consisting of some combination of the function name and its parameters—to enable type-safe linkage. Also called <i>name decoration</i> .
namespace	A mechanism that controls the scope of global names by allowing the global space to be divided into uniquely named scopes.
operator overloading	The ability to use the same operator notation to produce different outcomes. A special form of function overloading.
optimization	The process of improving the efficiency of the object code generated by the compiler.
options file	A user-provided file containing options desired for the compilation of templates, as well as source location and other information.
overloading	To give the same name to more than one function or operator.
polymorphism	The ability of a pointer or reference to refer to objects whose dynamic type is different from the declared pointer or reference type.
pragma	A compiler preprocessor directive, or special comment, that instructs the compiler to take a specific action.

runtime type identification (RTTI)	A mechanism that provides a standard method for a program to determine an object type during runtime.
rvalue	The variable located to the right of an assignment operator. The rvalue may be read but not altered.
stab	A symbol table entry generated in the object code. The same format is used in both <code>a.out</code> files and ELF files to contain debugging information.
stack	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.
static binding	Connection of a function call to a function body at compile time. Also called <i>early binding</i> .
subroutine	A function. In Fortran, a function that does not return a value.
symbol	A name or label that denotes some program entity.
symbol table	A list of all identifiers present when a program is compiled, their locations in the program, and their attributes. The compiler uses this table to interpret uses of identifiers.
template database	A directory containing all configuration files needed to handle and instantiate the templates required by a program.
template specialization	A specialized instance of a class template member function that overrides the default instantiation when the default cannot handle a given type adequately.
trapping	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.
type	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, structures, or by adding modifiers such as <code>pointer-to</code> or <code>constant</code> attributes.
VTABLE	A table created by the compiler for each class that contains virtual functions.

Index
