What's New in Sun WorkShop 6 update 2

Forte Developer 6 update 2 (Sun WorkShop 6 update 2)

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 806-7982-10
July 2001, Revision A

Send comments about this document to: docfeedback@sun.com

Please
Recycle

Adobe PostScript

Important Note on New Product Names

As part of Sun's new developer product strategy, we have changed the names of our development tools from Sun WorkShop™ to Forte™ Developer products. The products, as you can see, are the same high-quality products you have come to expect from Sun; the only thing that has changed is the name.

We believe that the Forte™ name blends the traditional quality and focus of Sun's core programming tools with the multi-platform, business application deployment focus of the Forte tools, such as Forte Fusion™ and Forte™ for Java™. The new Forte organization delivers a complete array of tools for end-to-end application development and deployment.

For users of the Sun WorkShop tools, the following is a simple mapping of the old product names in WorkShop 5.0 to the new names in Forte Developer 6.

Old Product Name	New Product Name
Sun Visual WorkShop™ C++	Forte™ C++ Enterprise Edition 6
Sun Visual WorkShop™ C++ Personal Edition	Forte™ C++ Personal Edition 6
Sun Performance WorkShop™ Fortran	Forte™ for High Performance Computing 6
Sun Performance WorkShop™ Fortran Personal Edition	Forte™ Fortran Desktop Edition 6
Sun WorkShop Professional™ C	Forte™ C 6
Sun WorkShop™ University Edition	Forte™ Developer University Edition 6

In addition to the name changes, there have been major changes to two of the products.

- Forte for High Performance Computing contains all the tools formerly found in Sun Performance WorkShop Fortran and now includes the C++ compiler, so High Performance Computing users need to purchase only one product for all their development needs.
- Forte Fortran Desktop Edition is identical to the former Sun Performance WorkShop Personal Edition, except that the Fortran compilers in that product no longer support the creation of automatically parallelized or explicit, directive-based parallel code. This capability is still supported in the Fortran compilers in Forte for High Performance Computing.

We appreciate your continued use of our development products and hope that we can continue to fulfill your needs into the future.

Contents

Before You Begin

What's New in Sun WorkShop 6 update 2 describes the new features of the Sun WorkShop™ 6 update 2 compilers and tools (plus the new features in the Sun WorkShop 6 and Sun WorkShop 6 update 1 releases).

Typographic Conventions

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

Shell Prompts

Shell	Prompt
C shell	%
Bourne shell and Korn shell	$
C shell, Bourne shell, and Korn shell superuser	#

Supported Platforms

This Sun WorkShop™ release supports versions 2.6, 7, and 8 of the Solaris™ *SPARC™ Platform Edition* and Solaris™ *Intel Platform Edition* operating environments.

Accessing Sun WorkShop Development Tools and Man Pages

The Sun WorkShop product components and man pages are not installed into the standard `/usr/bin/` and `/usr/share/man` directories. To access the Sun WorkShop compilers and tools, you must have the Sun WorkShop component directory in your `PATH` environment variable. To access the Sun WorkShop man pages, you must have the Sun WorkShop man page directory in your `MANPATH` environment variable.

For more information about the `PATH` variable, see the `csh`(1), `sh`(1), and `ksh`(1) man pages. For more information about the `MANPATH` variable, see the `man`(1) man page. For more information about setting your `PATH` and `MANPATH` variables to access this release, see the *Sun WorkShop 6 update 2 Installation Guide* or your system administrator.

Note – The information in this section assumes that your Sun WorkShop 6 update 2 products are installed in the `/opt` directory. If your product software is not installed in the `/opt` directory, ask your system administrator for the equivalent path on your system.

Accessing Sun WorkShop Compilers and Tools

Use the steps below to determine whether you need to change your `PATH` variable to access the Sun WorkShop compilers and tools.

To Determine If You Need to Set Your `PATH` Environment Variable

1. **Display the current value of the `PATH` variable by typing:**

```
% echo $PATH
```

2. **Review the output for a string of paths containing `/opt/SUNWspro/bin/`.**

   If you find the path, your `PATH` variable is already set to access Sun WorkShop development tools. If you do not find the path, set your `PATH` environment variable by following the instructions in the next section.

To Set Your `PATH` Environment Variable to Enable Access to Sun WorkShop Compilers and Tools

1. **If you are using the C shell, edit your home `.cshrc` file. If you are using the Bourne shell or Korn shell, edit your home `.profile` file.**

2. **Add the following to your `PATH` environment variable.**

   `/opt/SUNWspro/bin`

Accessing Sun WorkShop Man Pages

Use the following steps to determine whether you need to change your `MANPATH` variable to access the Sun WorkShop `man` pages.

To Determine If You Need to Set Your MANPATH Environment Variable

1. **Request the `workshop` man page by typing:**

```
% man workshop
```

2. **Review the output, if any.**

   If the `workshop`(1) man page cannot be found or if the man page displayed is not for the current version of the software installed, follow the instructions in the next section for setting your `MANPATH` environment variable.

To Set Your MANPATH Environment Variable to Enable Access to Sun WorkShop Man Pages

1. **If you are using the C shell, edit your home `.cshrc` file. If you are using the Bourne shell or Korn shell, edit your home `.profile` file.**

2. **Add the following to your `MANPATH` environment variable.**

   `/opt/SUNWspro/man`

Accessing Sun WorkShop Documentation

You can access Sun WorkShop product documentation at the following locations:

- **The product documentation is available from the documentation index installed with the product on your local system or network.**

  Point your Netscape™ Communicator 4.0 or compatible Netscape version browser to the following file:

  `/opt/SUNWspro/docs/index.html`

  If your product software is not installed in the `/opt` directory, ask your system administrator for the equivalent path on your system.

- **Manuals are available from the docs.sun.com℠ Web site.**

The `docs.sun.com` Web site (`http://docs.sun.com`) enables you to read, print, and buy Sun Microsystems manuals through the Internet. If you cannot find a manual, see the documentation index installed with the product on your local system or network.

Accessing Related Documentation

The following table describes related documentation that is available through the `docs.sun.com` Web site.

Document Collection	Document Title	Description
Numerical Computation Guide Collection	*Numerical Computation Guide*	Describes issues regarding the numerical accuracy of floating-point computations.
Solaris 8 Reference Manual Collection	See the titles of man page sections.	Provides information about the Solaris operating environment.
Solaris 8 Software Developer Collection	*Linker and Libraries Guide*	Describes the operations of the Solaris link-editor and runtime linker.
Solaris 8 Software Developer Collection	*Multithreaded Programming Guide*	Covers the POSIX and Solaris threads APIs, programming with synchronization objects, compiling multithreaded programs, and finding tools for multithreaded programs.

Ordering Sun Documentation

You can order product documentation directly from Sun through the `docs.sun.com` Web site or from Fatbrain.com, an Internet bookstore. You can find the Sun Documentation Center on Fatbrain.com at the following URL:

`http://www.fatbrain.com/documentation/sun`

Sending Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Email your comments to Sun at this address:

`docfeedback@sun.com`

CHAPTER 1

Sun WorkShop 6 update 2 New Features

This chapter describes the new features of the Sun WorkShop™ 6 update 2 compilers and tools. The primary focus of this release is improved performance, acceptance of non-standard C++ source code, and broadened OpenMP support.

This chapter has the following sections:

Documentation

To access the Sun WorkShop manuals described in this chapter, see "Accessing Sun WorkShop Documentation" on page 4 of this book.

For a description of the new and reorganized manuals for this release, see Chapter 2 in the *About Sun WorkShop 6 update 2 Documentation* manual. For a complete list of Sun WorkShop 6 update 2 book titles and descriptions, see Chapter 3 in the *About Sun WorkShop 6 update 2 Documentation* manual.

C Compiler

For more information about each of the new features listed in the following table, see the *C User's Guide* (each option has a listing in the book's index).

TABLE 1-1 C Compiler New Features

Feature	Option or Macro	Description
OpenMP	`-xopenmp`	Enables recognition of the pragmas listed in OpenMP C and C++ Application Program Interface Version 1.0 - October 1998. This specification is available from `http://www.openmp.org`.
Partial Support of C99	`-xc99`	Enables the compiler to accept the syntax and semantics for some of the features in the C99 standard.
`lint` Support for C99 Features	`-Xc99`	Enables `lint` to check whether your code correctly calls supported C99 features.
`lint` Support of Type-Based Alias Analysis	`-Xalias_level`	You can use `lint` to check how closely your code adheres to the level of type-based alias analysis you plan to specify at compile time.
Interprocedural Optimization	`-xipo`	Performs whole-program optimizations by invoking an interprocedural analysis pass. Unlike `-xcrossfile`, `-xipo` performs optimizations across all object files in the link step and is not limited to only the source files on the compile command.
Code Set Independence Support	`-xcsi`	Enables the C compiler to accept source code written in locales that do not conform to the ISO C source character code requirements. These locales include `ja_JP.PCK`.

TABLE 1-1 C Compiler New Features *(Continued)*

Feature	Option or Macro	Description
Identify Whether Char Types Are Signed	`-xchar`	Use this option only if you are migrating code from systems that define a char as unsigned. Do not use this option if your code does not rely on the signedness of char. Such code does not need to be rewritten to explicitly specify signed or unsigned char.

C++ Compiler

The table below lists the new features available with the Sun WorkShop 6 update 2 release of the C++ compiler (5.3). Some of the features are further described in the sections that follow the table, and all of the features are described in the *C++ User's Guide*.

TABLE 1-2 C++ Compiler New Features

Feature	Option	Description
Standard Iostreams Version of the Tools.h++ Library	`-library=rwtools7_std`	A version of the Tools.h++ library that works with standard iostreams is now available.
Shared `libCstd`		You can now dynamically link `libCstd`.
Shared `libiostream`		You can now dynamically link `libiostream`.
Performance Improvements		Several performance improvements have been made since the release of the Sun WorkShop 6 update 1 C++ compiler (5.2).
More Control Over Anachronism Warnings	`-features=[no%]transitions`	Some anachronism warnings can be turned into errors.
Acceptance of Nonstandard Source Code	`-features=[no%]extensions`	This new option enables you to compile nonstandard code that is commonly accepted by other C++ compilers.

TABLE 1-2 C++ Compiler New Features *(Continued)*

Feature	Option	Description
The `-xinline` Option Has Been Reinstated With a New Argument	`-xinline`	The `-xinline` option has been reinstated. In addition, it takes a new `%auto` argument, which enables automatic inlining at optimization levels of `-xO4` or higher.
Better Optimization of Standard Library Calls	`-xbuiltin`	The new `-xbuiltin` option directs the compiler to recognize as many of the built-in standard functions as possible.
Enhanced `-fast` Option	`-fast`	The `-fast` option now expands to include the `-xbuiltin=%all` option.
Interprocedural Optimization	`-xipo`	The new `-xipo` option performs whole-program optimizations by invoking an interprocedural analysis pass.
Requesting License Information as Part of Compile Command	`-xlicinfo`	Now you can compile and get license information in the same command.
Variable-Argument Preprocessor Function-Like Macros		The C++ compiler now supports preprocessor function macros that can take a variable number of arguments as defined in the C99 standard.
Trailing Comma in `enum` Declaration Is No Longer Treated as an Error		The compiler now accepts trailing commas in `enum` declarations and issues a warning.
New Option Chooses Handling of Nonlocal Static Object Initializers	`-features=[no%]split_init`	Use this option to specify whether to put all the initializers in one function or into separate functions (default).

Standard Iostreams Version of the Tools.h++ Library

This release of the C++ compiler contains two versions of the Tools.h++ library:

- **One that works with classic iostreams.** This version of the Tools.h++ library is compatible with the Tools.h++ library that was shipped with earlier versions of the compiler.
  - To use the classic iostreams version of Tools.h++ in standard mode (the default mode), use the `-library=rwtools7,iostream` option.
  - To use the classic iostreams version of Tools.h++ in compatibility mode (`-compat[=4]`), use the `-library=rwtools7` option.
- **One that works with standard iostreams.** This version of the Tools.h++ library is incompatible with the classic iostreams version of Tools.h++. This version is available only in standard mode and is not available in compatibility mode (`-compat[=4]`).

  To use the standard iostreams version of the library, use the `-library=rwtools7_std` option.

Shared `libCstd`

The Sun WorkShop 6 update 2 C++ compiler (5.3) includes a shared version of the `libCstd` library.

To use the shared version of `libCstd`, do the following:

1. **As superuser, manually create the following symbolic links.**

```
example% ln -s /usr/lib/libCstd.so.1 \
  /opt/SUNWSpro/lib/libCstd.so
example% ln -s /usr/lib/libCstd.so.1 \
  /opt/SUNWspro/lib/v8plus/libCstd.so
example% ln -s /usr/lib/sparcv9/libCstd.so.1 \
  /opt/SUNWSpro/lib/v9/libCstd.so
```

For the Intel 32-bit processor architecture, you do not need the last two links.

Note – If your compiler is not installed in the `/opt/SUNWSpro` directory, ask your system administrator for the equivalent path on your system.

---

2. **Test the links.**

Compile any program with `/opt/SUNWSpro/bin/CC`, then type the command `ldd a.out`. The output shows any dependency on `/usr/lib/libCstd.so.1`.

Once these symbolic links are created, `libCstd` is linked dynamically by default.

To link `libCstd` statically, use the `-staticlib=Cstd` option.

If you plan to distribute object files that were linked with the shared `libCstd` library you must do one of the following:

- Distribute the latest `SUNWlibC` patch with your product

  or

- Require your customers to download the latest `SUNWlibC` patch from a Sun Web site, such as `http://sunsolve.sun.com`.

Shared `libiostream`

The Sun WorkShop 6 update 2 C++ compiler (5.3) includes a shared version of the classic iostream library `libiostream`.

To use the shared version of `libiostream`, do the following:

1. **As superuser, manually create the following symbolic links.**

```
example% ln -s /usr/lib/libiostream.so.1 \
  /opt/SUNWspro/lib/libiostream.so
example% ln -s /usr/lib/sparcv9/libiostream.so.1 \
  /opt/SUNWSpro/lib/v9/libiostream.so
```

For the Intel 32-bit processor architecture, you do not need the last link.

Note – If your compiler is not installed in the `/opt/SUNWSpro` directory, ask your system administrator for the equivalent path on your system.

---

2. **Test the links.**

Compile any program with `/opt/SUNWSpro/bin/CC` using the option `-library=iostream`. Next, type the command `ldd a.out`. The output shows a dependency on `/usr/lib/libiostream.so.1`.

Once these symbolic links are created, the compiler dynamically links `libiostream` by default whenever you use `-library=iostream`.

To link this library statically, use the `-library=iostream -staticlib=iostream` options.

If you plan to distribute object files that were linked with the shared `libiostream` library you must do one of the following:

- Distribute the latest `SUNWlibC` patch with your product

  or

- Require your customers to download the latest `SUNWlibC` patch from a Sun Web site, such as `http://sunsolve.sun.com`.

Performance Improvements

The Sun WorkShop 6 update 2 C++ (5.3) compiler has the following performance improvements (as compared to the 5.2 C++ compiler):

- Significant improvement in compile time, especially for parallel compilations in the same directory that have heavy use of templates.
- Better performance of the standard library, particularly iostreams and the string class. String class improvements rely on special code generation when you select the `-xarch=v8plus` or `-xarch=v9` option.
- Better management of inline function generation. Some functions are now inlined that previously were considered to exceed the compiler's complexity measure. Default inlining is now tuned to reduce compilation time at low levels of optimization and to improve runtime speed at high levels of optimization.

More Control Over Anachronism Warnings

In compatibility mode (`-compat[=4]`), the compiler is silent about transition errors. The compiler emits warnings about these errors when you specify `+w` or `+w2`.

In standard mode, the compiler emits warnings about transition errors.

When you use the new `-features=no%transitions` option in either compatibility mode or standard mode, the compiler emits error messages instead of warnings.

The new `-features=no%transitions` option controls the following language constructs:

- Redefining a template after it is used
- Omitting the `typename` directive when it is needed in a template definition
- Implicitly declaring type `int`

For more information, see the *C++ User's Guide*.

Acceptance of Nonstandard Source Code

The new `-features=extensions` option enables you to compile nonstandard code that is commonly accepted by other C++ compilers.

You can use this option when you must compile invalid code (when you are not permitted to modify the code to make it valid).

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

When you add `-features=extensions` to a compile command, the compiler supports the following language extensions:

- Overriding virtual functions can be less restrictive than the functions that they override.
- Forward declarations of `enum` types and variables are allowed.
- Incomplete `enum` types are taken as forward declarations.
- `enum` name is allowed as a scope qualifier.
- Anonymous `struct` declarations are allowed.
- Passing the address of an anonymous class instance is allowed.
- A static namespace-scope function is allowed as a class friend.
- The predefined `__func__` symbol for function name is allowed.

For more information, see the *C++ User's Guide.*

Fortran Compilers

Sun WorkShop 6 update 2 provides both Fortran 95 (`f95`) and Fortran 77 (`f77`) compilers. The table below lists the new features in the Sun WorkShop 6 update 2 release of the Fortran 95 compiler. There are no new features released for Fortran 77. See the *Fortran User's Guide* for details.

TABLE 1-3 Fortran 95 Compiler New Features

Feature	Description
VAX Structures	To aid migration of programs from `f77`, `f95` accepts VAX Fortran `STRUCTURE` and `UNION` statements, which are precursors of Fortran 95 derived types. See Appendix C in the *Fortran User's Guide*.
Extended `ALLOCATABLE` Attribute	Recent decisions by the Fortran 95 standards organizations have extended the data entities that are allowed for the `ALLOCATABLE` attribute. Previously limited to local arrays, the `ALLOCATABLE` attribute is now allowed with array components of structures, dummy arrays, and array function results. See Appendix C in the *Fortran User's Guide*.
`VALUE` Attribute	`f95` recognizes the `VALUE` attribute, a feature that is proposed for the Fortran 2000 standard. Declaring a subprogram dummy argument with `VALUE` indicates that the actual argument is passed *by value* rather than *by reference*. See Appendix C in the *Fortran User's Guide*.
Stream I/O	Another feature proposed for Fortran 2000 is a new stream I/O scheme that treats a data file as a continuous sequence of bytes that are addressable by a positive integer starting with 1. Enable stream I/O by declaring a file with `ACCESS='STREAM'`. Position files with a `READ` or `WRITE` statement with the `POS=`*integer_expression* specifier. See Appendix C in the *Fortran User's Guide*.
Global Program Checking	Invoked by the `-Xlist` options, global program checking on `f95` now resembles `f77` and includes the suboptions `-Xlistc`, `-Xlisth`, `-Xlists`, `-Xlistv`*n*, and `-Xlistw[`*n*`]`. See Chapter 3 in the *Fortran User's Guide*.
Fortran Library Interface	`f95` recognizes the include file `system.inc` for declaring the proper data types for the Fortran library. Supply the statement `INCLUDE 'system.inc'` in every routine that references non-intrinsic Fortran library routines to ensure proper typing of return values.
Interprocedural Optimization	With the new `-xipo` option, the compiler performs whole-program optimizations (subprogram inlining). Unlike the `-xcrossfile` option, `-xipo` does not require all the files to be on a single compile command. `-xipo` is useful when compiling and linking large multiple-file applications. See Chapter 3 in the *Fortran User's Guide*.

TABLE 1-3 Fortran 95 Compiler New Features *(Continued)*

Feature	Description
OpenMP 2.0 Enhancements	`f95` now supports the OpenMP 2.0 Fortran API. Enhancements include `WORKSHARE`, `REDUCTION` for arrays, `THREADPRIVATE` for variables, and `COPYPRIVATE` for `SINGLE` directives. See Appendix E in the *Fortran User's Guide* and the `http://www.openmp.org` Web site.
OpenMP Library Interface	The compiler now provides an include file and an interface module for defining the interfaces to the OpenMP Fortran library routines. Supply either of the following statements in every program unit that references the OpenMP library to ensure proper type declarations: `INCLUDE 'omp_lib.h'` or `USE omp_lib`. See Appendix E in the *Fortran User's Guide*.

dbx

Some of the new features in `dbx` are described in greater detail in the sections following the table.

TABLE 1-4 `dbx` New Features

Feature	Description
Elimination of 8-Megabyte Limit on Runtime Checking	The 8-megabyte limit on runtime checking no longer applies on hardware based on the UltraSPARC processors, on which `dbx` has the ability to invoke a trap handler instead of using a branch.
Debugging a Mismatched Core File	Preliminary support for the debugging of "mismatched" core files (for example, core files that are produced on a system that has a different version or patch level of the Solaris operating environment) is available.
`step to` Command	The new `step to` command attempts to step into the specified function in the current source code line.
Support for `gdb` Commands	The `gbd` command supports the `gdb` command set.
Changes to `collector` Commands	The `collector pause` and `collector resume` commands are new. The `collector enable_once` command has been removed. The `collector store` command has been extended to include experiment groups. For more information, see "`collector` Command" in Appendix C of *Debugging a Program With* `dbx` and the `collector`(1) man page.

TABLE 1-4 dbx New Features *(Continued)*

Feature	Description
Fortran Intrinsics Support Removed on Intel Platforms	Fortran intrinsics support on Intel platforms has been removed.

Elimination of 8-Megabyte Limit on Runtime Checking

The 8-megabyte limit on runtime checking no longer applies on hardware based on the UltraSPARC™ processors, on which `dbx` has the ability to invoke a trap handler instead of using a branch. The transfer of control to a trap handler is up to ten (10) times slower but is not subject to the 8-megabyte limit. Traps are used automatically, as necessary, as long as the hardware is based on UltraSPARC processors. To check your hardware, use the system command `isalist` and check that the result contains the string `sparcv8plus`.

The `dbx` environment variable `rtc_use_traps` has been removed.

Debugging a Mismatched Core File

To determine where your program is crashing, you might want to examine the core file, which is the memory image of your program when it crashed. If the program that dumped core was dynamically linked with any shared libraries, it is best to debug the core file in the same operating environment in which it was created. However, it is not always possible or convenient to do so. `dbx` now has limited support for the debugging of "mismatched" core files (for example, core files that are produced on a system that has a different version or patch level of the Solaris operating environment).

Two problems with libraries might arise when you debug a "mismatched" core file:

- The shared libraries that are used by the program on the core-host will not be the same libraries that are used on the `dbx`-host. To get accurate stack traces involving the libraries, you need to make these original libraries available on the `dbx`-host.
- `dbx` uses system libraries in the `/usr/lib` directory to help understand the implementation details of the runtime linker and threads library on the system. You might need to provide these system libraries from the core-host so that `dbx` can understand the runtime linker data structures and the threads data structures.

For information on eliminating these library problems and debugging a "mismatched" core file with `dbx`, see "Debugging a Mismatched Core File" in Chapter 2 of *Debugging a Program With dbx*.

`step to` Command

The `step to` command attempts to step into the specified function in the current source code line. If no function is specified, the command attempts to step into the last function that was called as determined by the assembly code for the current source code line.

The function call might not be taken due to a conditional branch. In the case where the call is not taken or there is no function call in the current source code line, the `step to` command steps over the current source code line.

For complete information on the `step to` command, see "`step` Command" in Appendix C of *Debugging a Program With* `dbx`.

Support for `gdb` Commands

The `gdb` command provides support for the `gdb` command set. Use the `gdb on` command to enter the `gdb` command mode, in which `dbx` understands and accepts `gdb` commands. Use the `gdb off` command to exit `gdb` command mode and return to `dbx` command mode. `dbx` commands are not accepted in `gdb` command mode, and `gdb` commands are not accepted in `dbx` command mode. All debugging settings (such as breakpoints) are preserved across different command modes.

The following `gdb` commands are not supported in this release:

- `command`
- `define`
- `handle`
- `hbreak`
- `interrupt`
- `maintenance`
- `printf`
- `rbreak`
- `return`
- `signal`
- `tcatch`
- `until`

Sun WorkShop TeamWare

TABLE 1-5 Sun WorkShop TeamWare New Feature

Feature	Description
Quick Tour Accessible From the Help Menu	The interactive Quick Tour, which provides a high-level overview of Sun WorkShop TeamWare's basic model and features, is now accessible from the Help menu (choose Help ➤ TeamWare Quick Tour) in any Sun WorkShop TeamWare window.

Sun Performance Library

Sun Performance Library™ is a set of optimized, high-speed mathematical subroutines for solving linear algebra problems and other numerically intensive problems. Sun Performance Library is based on a collection of public domain applications available from Netlib (at `http://www.netlib.org`). These routines have been enhanced and bundled as the Sun Performance Library.

TABLE 1-6 Sun Performance Library New Features

Feature	Description
Performance Increases	Performance has been increased for the following: • LU factorization routines (`GETRF`) • Long vector complex and double complex 1D FFT routines (where long vector is defined as power-of-2 vectors of length > 131072) • Selected BLAS routines • Sparse solver numeric factorization and solve routines
Additional Routines Parallelized	Long vector complex and double complex 1D FFT routines have been parallelized.
Sparse Solver Enhancements	Additional sparse matrix ordering methods have been added to the sparse solver to reduce memory usage and shorten runtimes, depending on the sparse system.

Announcement to Remove LINPACK From Future Versions of Sun Performance Library

After the Sun WorkShop 6 update 2 release of the Sun Performance Library, LINPACK will no longer be included in the library. LAPACK version 3.0 supersedes LINPACK and all previous versions of LAPACK. If legacy user codes that call LINPACK routines cannot be modified to use LAPACK routines, the public domain version of LINPACK can still be obtained from Netlib.

TABLE 1-7 LINPACK Routines to Be Removed in the Next Release

CCHDC	DCHDC	SCHDC	ZCHDC	CCHDD	DCHDD	SCHDD	ZCHDD
CCHEX	DCHEX	SCHEX	ZCHEX	CCHUD	DCHUD	SCHUD	ZCHUD
CGBCO	DGBCO	SGBCO	ZGBCO	CGBDI	DGBDI	SGBDI	ZGBDI
CGBFA	DGBFA	SGBFA	ZGBFA	CGBSL	DGBSL	SGBSL	ZGBSL
CGECO	DGECO	SGECO	ZGECO	CGEDI	DGEDI	SGEDI	ZGEDI
CGEFA	DGEFA	SGEFA	ZGEFA	CGESL	DGESL	SGESL	ZGESL
CGTSL	DGTSL	SGTSL	ZGTSL	CHICO	ZHICO	CHIDI	ZHIDI
CHIFA	ZHIFA	CHISL	ZHISL	CHPCO	ZHPCO	CHPDI	ZHPDI
CHPFA	ZHPFA	CHPSL	ZHPSL	CPBCO	DPBCO	SPBCO	ZPBCO
CPBDI	DPBDI	SPBDI	ZPBDI	CPBFA	DPBFA	SPBFA	ZPBFA
CPBSL	DPBSL	SPBSL	ZPBSL	CPOCO	DPOCO	SPOCO	ZPOCO
CPODI	DPODI	SPODI	ZPODI	CPOFA	DPOFA	SPOFA	ZPOFA
CPOSL	DPOSL	SPOSL	ZPOSL	CPPCO	DPPCO	SPPCO	ZPPCO
CPPDI	DPPDI	SPPDI	ZPPDI	CPPFA	DPPFA	SPPFA	ZPPFA
CPPSL	DPPSL	SPPSL	ZPPSL	CPTSL	DPTSL	SPTSL	ZPTSL
CQRDC	DQRDC	SQRDC	ZQRDC	CQRSL	DQRSL	SQRSL	ZQRSL
CSICO	DSICO	SSICO	ZSICO	CSIDI	DSIDI	SSIDI	ZSIDI
CSIFA	DSIFA	SSIFA	ZSIFA	CSISL	DSISL	SSISL	ZSISL
CSPCO	DSPCO	SSPCO	ZSPCO	CSPDI	DSPDI	SSPDI	ZSPDI
CSPFA	DSPFA	SSPFA	ZSPFA	CSPSL	DSPSL	SSPSL	ZSPSL
CSVDC	DSVDC	SSVDC	ZSVDC	CTRCO	DTRCO	STRCO	ZTRCO
CTRDI	DTRDI	STRDI	ZTRDI	CTRSL	DTRSL	STRSL	ZTRSL

Performance Analyzer

For a complete description of the new features described in the following table, see the *Analyzing Program Performance With Sun WorkShop* manual and the `analyzer`(1), `collect`(1), `collector`(1), `dbx`(1), `er_print`(1), `er_src`(1) and `libcollector`(3) man pages.

TABLE 1-8 Performance Analyzer New Features

Feature	Description
Dual Hardware Counter Overflow Experiments	Data for two hardware counters can be collected in the same experiment provided the counters use different registers.
Compiler Commentary	Compiler commentary now appears in annotated source code. A new standalone source browser, `er_src`, is available for viewing compiler commentary without needing to run the Samping Collector and Performance Analyzer.
Pausing and Resuming Data Collection	Data collection can be paused and resumed during the course of an experiment. This feature is available with the `collect` command using signals, with the `dbx collector` command using the `pause` and `resume` subcommands, and from the `libcollector` API.
MPI Experiment Names	The default name for an MPI experiment includes the MPI rank.
Experiment Name Standards	Experiment names must end in `.er`, and experiment group names must end in `.erg`.
Experiment Group Support	Experiment groups are now supported in `dbx` with the `dbx collector store group` subcommand.
Defaults Files	The Performance Analyzer and the `er_print` command read defaults files named `.er.rc` in which defaults for metrics and other features can be included.
Hardware Counter Metric Name Changes	The names of some of the hardware counter metrics have been changed.
Obsolete `dbx` Commands	The `collector enable_once`, `collector close`, and `collector quit` commands are no longer supported.

CHAPTER 2

Sun WorkShop 6 update 1 New Features

This chapter describes the new features of the Sun WorkShop™ 6 update 1 compilers and tools. The primary focus of this release is improved performance on the UltraSPARC™ III processor.

Each updated Sun WorkShop component has a section that includes a table that summarizes the new features. Explanations of some new features follow the summary tables.

This chapter has the following sections:

C Compiler

TABLE 2-1 lists the new features available with the Sun WorkShop 6 update 1 release of the C compiler. Some of the features are described in greater detail in the sections following the table.

TABLE 2-1 C Compiler New Features

Feature	Option or Macro	Description
Support for the UltraSPARC III Processor	`-xtarget` `-xchip`	The `-xtarget` and `-xchip` options now accept `ultra3`. See the following discussion for the recommended flags to use for optimal UltraSPARC III performance.
Optimizing Through Type-Based Analysis	`-xalias_level`	The C compiler now accepts options and pragmas which allow it to perform type-based alias analysis and optimizations.
Enhancing Math Routine Performance With New Pragmas	`__MATHERR_ERRNO_DONTCARE`	`cc -fast` now expands to include the macro `__MATHERR_ERRNO_DONTCARE`. This macro causes `math.h` in the Solaris 8 operating environment to assert performance-related pragmas for some of the math routines prototyped in `math.h`.
Inlining Standard Library Functions	`-xbuiltin`	Use this command to improve performance of generated code through substitution of intrinsics, or inlining, of standard library functions where the compiler determines it is profitable.

TABLE 2-1 C Compiler New Features *(Continued)*

Feature	Option or Macro	Description
Enabling and Disabling Trigraph Recognition	`-xtrigraphs`	The new `-xtrigraphs` option enables and disables trigraph translation.
Prefetch Latency Specifier	`-xprefetch=...,latx:`*factor*	This new suboption determines how far apart the instruction the compiler generates to prefetch data will be from the subsequent use of the data in a load or store instruction.
Overriding the Default Search Path With the `-I-` Option	`-I-`	The new `-I-` option gives you more control over the algorithm that the compiler uses when searching for include files.

Support for the UltraSPARC III Processor

Use the following options to compile for optimal performance when compiling and running your program on an UltraSPARC III processor:

```
-fast -xcrossfile -xprofile={collect:|use:}
```

- For cross-compilation (compiling on a platform other than UltraSPARC III but generating object binaries to run on an UltraSPARC III system), add the following options to insure correct cache sizes and optimization strategies:

  ```
  -xtarget=ultra3 -xarch={v8plusb|v9b}
  ```

- Specify `-xarch=v8plusb` for 32-bit code generation and `v9b` for 64-bit code. For programs accessing very large data files, 64-bit code provides better performance. However, `-xarch=v9b` should only be used when 64-bit code generation is required. In some cases it can result in slower performance.
- The `-fast` option enables optimizations that favor speed of execution and raises the optimization level to `-xO5`. Note that `-fast` allows substitution of faster arithmetic operations where possible (`-fsimple=2`), so some inaccuracies might result.
- `-xcrossfile` enables the compiler to apply optimizations across all the source files specified on the command line, including some interprocedural optimizations.
- `-xprofile={collect:|use:}` enables program performance profiling. Profiling allows the compiler to identify the most frequently executed sections of code and to perform localized optimizations to best advantage.

Note – Programs compiled specifically for the UltraSPARC III platform with `-xarch={v8plusb|v9b}` will not operate on platforms other than the UltraSPARC III platform. Use `-xarch={v8plusa|v9a}` to compile programs to run compatibly on UltraSPARC I, UltraSPARC II, and UltraSPARC III platforms.

Optimizing Through Type-Based Analysis

You can use the new `-xalias_level` C compiler command and several new pragmas to enable the compiler to perform type-based alias analysis and optimizations. Use these extensions to express type-based information about the way pointers are used in your C program. The C compiler uses this information, in turn, to do a significantly better job of alias disambiguation for pointer-based memory references in your program.

For more information on this new compiler command, see the *C User's Guide Supplement* available from `http://docs.sun.com` (in the Forte™ Developer 6 update 1/Sun WorkShop 6 update 1 Collection).

Enhancing Math Routine Performance With New Pragmas

Now when you issue the `-fast` option, the macro `__MATHERR_ERRNO_DONTCARE` is defined. This macro causes `math.h` in the Solaris 8 operating environment to assert performance-related pragmas such as the following for some math routines prototyped in `<math.h>`:

- `#pragma does_not_read_global_data`
- `#pragma does_not_write_global_data`
- `#pragma no_side_effect`

If your code relies on the return value of `errno` in exceptional cases as documented in the `matherr`(3M) man page, you must turn off the macro by issuing the `-U__MATHERR_ERRNO_DONTCARE` macro after the `-fast` option.

Inlining Standard Library Functions

Use the `-xbuiltin[=%all|%none]` command when you want to improve the optimization of code that calls standard library functions. Many standard library functions, such as the ones defined in `math.h` and `stdio.h`, are commonly used by

various programs. This command lets the compiler substitute intrinsic pure functions for system functions to improve performance. The `-xbuiltin` command has the following syntax:

- `-xbuiltin=%all`
- `-xbuiltin=%none`

The first default of this command is `-xbuiltin=%none`, which means no functions from the standard libraries are inlined. The first default applies when you do not specify `-xbuiltin`.

The second default is `-xbuiltin=%all`, which means the compiler inlines standard library functions as it determines the optimization benefit. The second default applies when you specify `-xbuiltin` but do not provide an argument.

If you compile with the `-fast` option, then `-xbuiltin` is set to `%all`.

Enabling and Disabling Trigraph Recognition

The `-xtrigraphs` option determines whether the compiler recognizes trigraph sequences as defined by the ISO/ANSI C standard.

By default, the compiler assumes `-xtrigraphs=yes` and recognizes all trigraph sequences throughout the compilation unit.

If your source code has a literal string containing question marks (?) that the compiler is interpreting as a trigraph sequence, you can use the `-xtrigraph=no` suboption to turn off the recognition of trigraph sequences. The `-xtrigraphs=no` option turns off recognition of all trigraphs throughout the entire compilation unit.

Consider the following example source file named `trigraphs_demo.c`.

```
#include <stdio.h>

int main ()
{

  (void) printf("(\?\?) in a string appears as (??)\n");

  return 0;
}
```

Here is the output if you compile this code with `-xtrigraphs=yes`.

```
example% cc -xtrigraphs=yes trigraphs_demo.c
example% a.out
(??) in a string appears as []
```

Here is the output if you compile this code with `-xtrigraphs=no`.

```
example% cc -xtrigraphs=no trigraphs_demo.c
example% a.out
(??) in a string appears as (??)
```

For more information on using the `-xtrigraphs` option, see the `cc`(1) man page. For information on trigraphs, see the *C User's Guide* chapter about transitioning to ANSI/ISO C.

Prefetch Latency Specifier

(*SPARC* platform) If you are running computationally intensive codes on large multiprocessors, you might find it advantageous to use the new `-xprefetch` suboption `latx:`*factor*. This suboption instructs the code generator to adjust the default latency time between a prefetch and its associated load or store by the specified factor.

For more information, see the `cc`(1) man page.

Overriding the Default Search Path With the `-I-` Option

The new `-I-` option gives you more control over the algorithm that the compiler uses when searching for include files. This section first describes the default search algorithms, then it describes the effect of `-I-` on these algorithms.

Default Search Algorithm for Quote-Included Files

For statements of the form `#include "foo.h"` (where quotation marks are used), the compiler searches for include files in the following order:

1. The current directory (that is, the directory containing the "including" file)

2. The directories named with `-I` options, if any
3. The `/usr/include` directory

Default Search Algorithm for Bracket-Included Files

For statements of the form `#include <foo.h>` (where angle brackets are used), the compiler searches for include files in the following order:

1. The directories named with `-I` options, if any
2. The `/usr/include` directory

Using the `-I-` Option to Change the Search Algorithm

The new `-I-` option gives more control over the default search rules. When `-I-` appears in the command line:

- The compiler never searches the current directory, unless the directory is listed explicitly in a `-I` directive. This effect applies even for include statements of the form `#include "foo.h"`.
- For include statements of the form `#include "foo.h"`, the compiler searches for include files in the following order:
  a. The directories named with `-I` options (both before and after `-I-`)
  b. The `/usr/include` directory
- For include statements of the form `#include <foo.h>`, the compiler searches for include files in the following order:
  a. The directories named with `-I` that appear after `-I-` (that is, the compiler does not search the `-I` directories that appear before `-I-`)
  b. The `/usr/include` directory

The following example shows the results of using `-I-` when compiling `prog.c`.

`prog.c`	`#include "a.h"`
	`#include <b.h>`
	`#include "c.h"`

`c.h`	`#ifndef _C_H_1`
	`#define _C_H_1`
	`int c1;`
	`#endif`

`int/a.h`	`#ifndef _A_H`
	`#define _A_H`
	`#include "c.h"`
	`int a;`
	`#endif`

`int/b.h`	`#ifndef _B_H`
	`#define _B_H`
	`#include <c.h>`
	`int b;`
	`#endif`

`int/c.h`	`#ifndef _C_H_2`
	`#define _C_H_2`
	`int c2;`
	`#endif`

The following command shows the default behavior of searching the current directory (the directory of the including file) for include statements of the form `#include "foo.h"`. When processing the `#include "c.h"` statement in `inc/a.h`, the preprocessor includes the `c.h` header file from the `inc` subdirectory. When processing the `#include "c.h"` statement in `prog.c`, the preprocessor includes the `c.h` file from the directory containing `prog.c`. Note that the `-H` option instructs the compiler to print the paths of the included files.

```
example% cc -c -Iinc -H prog.c
inc/a.h
        inc/c.h
inc/b.h
        inc/c.h
c.h
```

The next command shows the effect of the `-I-` option. The preprocessor does not look in the including directory first when it processes statements of the form `#include "foo.h"`. Instead, it searches the directories named by the `-I` options in the order that they appear in the command line. When processing the `#include "c.h"` statement in `inc/a.h`, the preprocessor includes the `./c.h` header file instead of the `inc/c.h` header file.

```
example% cc -c -I. -I- -Iinc -H prog.c
inc/a.h
        ./c.h
inc/b.h
        inc/c.h
./c.h
```

For more information, see the entry for `-I-` in the `cc`(1) man page.

C++ Compiler

TABLE 2-2 lists the new features available with the Sun WorkShop 6 update 1 release of the C++ compiler (5.2). Some of the features are described in greater detail in the sections following the table.

TABLE 2-2 C++ Compiler New Features

Feature	Options	Description
Support for the UltraSPARC III Processor	`-xtarget` `-xchip`	The `-xtarget` and `-xchip` options now accept `ultra3`. See the following discussion for the recommended flags to use for optimal UltraSPARC III performance.
Compile-Time Performance		The compiler is substantially faster for many large programs, particularly those that use templates heavily.

TABLE 2-2 C++ Compiler New Features *(Continued)*

Feature	Options	Description
Lifetime of Temporary Objects	`-features=tmplife`	(Standard mode only) A new `-features=tmplife` suboption instructs the compiler to destroy temporary objects according to the requirements of the C++ standard.
Overriding the Default Search Path With the `-I-` Option	`-I-`	The new `-I-` option gives you more control over the algorithm that the compiler uses when searching for include files.
Interval Arithmetic Support for C++	`-xia` `-library=[no%]interval`	(*SPARC* platform) This release of the C++ compiler provides a C++ interface to the interval arithmetic library.
Mixed-Language Linking	`-xlang` `-staticlib`	The new `-xlang` option enables linking of mixed Fortran and C++ object files.
Enabling and Disabling Trigraph Recognition	`-xtrigraphs`	The new `-xtrigraphs` option enables and disables trigraph translation.
Filtering Linker Error Messages	`-filt`	The new `-filt` option allows you to customize the filtering of linker error messages. For example, you can request mangled names.
Shared `libCstd`		A shared version of `libCstd` is available in the `lib` directory of the C++ compiler.
Shared `libiostream`		A shared version of `libiostream` is available in the `lib` directory of the C++ compiler.
Optimization Pragmas		The C++ compiler recognizes the new optimization pragmas `no_side_effects` and `returns_new_memory`.

TABLE 2-2 C++ Compiler New Features *(Continued)*

Feature	Options	Description
Recognition of `.c++` Extension		The C++ compiler now recognizes `.c++` as a valid filename suffix.
Prefetch Latency Specifier	`-xprefetch=...,latx:`*factor*	This new suboption determines how far apart the instruction the compiler generates to prefetch data will be from the subsequent use of the data in a load or store instruction.

Support for the UltraSPARC III Processor

Use the following options to compile for optimal performance when compiling and running your program on an UltraSPARC III processor:

```
-fast -xcrossfile -xprofile={collect:|use:}
```

- For cross-compilation (compiling on a platform other than UltraSPARC III but generating object binaries to run on an UltraSPARC III system), add the following options to insure correct cache sizes and optimization strategies:

  ```
  -xtarget=ultra3 -xarch={v8plusb|v9b}
  ```

- Specify `-xarch=v8plusb` for 32-bit code generation and `v9b` for 64-bit code. `-xarch=v9b` should only be used when 64-bit code generation is required. In some cases it can result in slower performance.
- The `-fast` option enables optimizations that favor speed of execution and raises the optimization level to `-xO5`. Note that `-fast` allows substitution of faster arithmetic operations where possible (`-fsimple=2`), so some inaccuracies might result.
- `-xcrossfile` enables the compiler to apply optimizations across all the source files specified on the command line, including some interprocedural optimizations.
- `-xprofile={collect:|use:}` enables program performance profiling. Profiling allows the compiler to identify the most frequently executed sections of code and to perform localized optimizations to best advantage.

Note – Programs compiled specifically for the UltraSPARC III platform with `-xarch={v8plusb|v9b}` will not operate on platforms other than the UltraSPARC III platform. Use `-xarch={v8plusa|v9a}` to compile programs to run compatibly on UltraSPARC I, UltraSPARC II, and UltraSPARC III platforms.

Lifetime of Temporary Objects

When the compiler evaluates expressions, it sometimes creates temporary objects. For example, the compiler may create a temporary object when a function is called or when a cast creates a class object. The old language definition allowed such temporary objects to be destroyed at any time up until the end of the block in which the temporary object was created. The C++ standard specifies that the compiler must destroy a temporary object at the end of the full expression in which the object is created, unless the object is used to initialize a reference.

By default, the C++ compiler (5.2) implements the old language definition; it destroys temporary objects at the end of the block in which they are created. To make the compiler destroy temporary objects according to the requirements of the C++ standard, use the new `-features=tmplife` suboption.

For example, consider the following block.

```
{  foo(ClassA()); bar(ClassB()); some-statement; }
```

If you specify `-features=tmplife` in the command line, the call sequence is as shown here.

```
tmp1=ClassA(); foo(tmp1); tmp1.~ClassA();
tmp2=ClassB(); bar(tmp2); tmp2.~ClassB();
some-statement;
```

If you do not specify the `-features=tmplife` suboption, the call sequence is as shown here.

```
tmp1=ClassA(); foo(tmp1);
tmp2=ClassB(); bar(tmp2);
some-statement;
tmp2.~ClassB();
tmp1.~ClassA();
```

This suboption is not available in compatibility mode (`-compat[=4]`).

Overriding the Default Search Path With the `-I-` Option

The new `-I-` option gives you more control over the algorithm that the compiler uses when searching for include files. This section first describes the default search algorithms, then it describes the effect of `-I-` on these algorithms.

Default Search Algorithm for Quote-Included Files

For statements of the form `#include "foo.h"` (where quotation marks are used), the compiler searches for include files in the following order:

1. The current directory (that is, the directory containing the "including" file)
2. The directories named with `-I` options, if any
3. The directories for compiler-provided C++ header files, ANSI C header files, and special-purpose files
4. The `/usr/include` directory

Default Search Algorithm for Bracket-Included Files

For statements of the form `#include <foo.h>` (where angle brackets are used), the compiler searches for include files in the following order:

1. The directories named with `-I` options, if any
2. The directories for compiler-provided C++ header files, ANSI C header files, and special-purpose files
3. The `/usr/include` directory

Note – If the name of the include file matches the name of a standard header, also refer to Section 5.7.4 Standard Header Implementation in the *C++ User's Guide*.

Using the `-I-` Option to Change the Search Algorithm

The new `-I-` option gives more control over the default search rules. When `-I-` appears in the command line:

- The compiler never searches the current directory, unless the directory is listed explicitly in a `-I` directive. This effect applies even for include statements of the form `#include "foo.h"`.
- For include statements of the form `#include "foo.h"`, the compiler searches for include files in the following order:

  a. The directories named with `-I` options (both before and after `-I-`)

  b. The directories for compiler-provided C++ header files, ANSI C header files, and special-purpose files

  c. The `/usr/include` directory
- For include statements of the form `#include <foo.h>`, the compiler searches for include files in the following order:

  a. The directories named with `-I` options that appear after `-I-` (that is, the compiler does not search the `-I` directories that appear before `-I-`)

  b. The directories for compiler-provided C++ header files, ANSI C header files, and special-purpose files

  c. The `/usr/include` directory

The following example shows the results of using `-I-` when compiling `prog.cc`.

`prog.cc`	`#include "a.h"`
	`#include <b.h>`
	`#include "c.h"`

`c.h`	`#ifndef _C_H_1`
	`#define _C_H_1`
	`int c1;`
	`#endif`

inc/a.h	#ifndef _A_H #define _A_H #include "c.h" int a; #endif
inc/b.h	#ifndef _B_H #define _B_H #include <c.h> int b; #endif
inc/c.h	#ifndef _C_H_2 #define _C_H_2 int c2; #endif

The following command shows the default behavior of searching the current directory (the directory of the including file) for include statements of the form `#include "foo.h"`. When processing the `#include "c.h"` statement in `inc/a.h`, the compiler includes the `c.h` header file from the `inc` subdirectory. When processing the `#include "c.h"` statement in `prog.cc`, the compiler includes the `c.h` file from the directory containing `prog.cc`. Note that the `-H` option instructs the compiler to print the paths of the included files.

```
example% CC -c -Iinc -H prog.cc
inc/a.h
        inc/c.h
inc/b.h
        inc/c.h
c.h
```

The next command shows the effect of the `-I-` option. The compiler does not look in the including directory first when it processes statements of the form `#include "foo.h"`. Instead, it searches the directories named by the `-I` options in the order

that they appear in the command line. When processing the `#include "c.h"` statement in `inc/a.h`, the compiler includes the `./c.h` header file instead of the `inc/c.h` header file.

```
example% CC -c -I. -I- -Iinc -H prog.cc
inc/a.h
        ./c.h
inc/b.h
        inc/c.h
./c.h
```

For more information, see the entry for `-I-` in the `CC`(1) man page.

Interval Arithmetic Support for C++

(*SPARC* platform) Sun WorkShop™ 6 update 1 Compilers C++ (5.2) provides a C++ interface to the C++ interval arithmetic library. For more information, see "Interval Arithmetic" on page 47 in this document and the *C++ Interval Arithmetic Programming Reference* available from `http://docs.sun.com` (in the Forte™ Developer 6 update 1/Sun WorkShop 6 update 1 Collection).

For information about related compiler options, see the entries for the following in the `CC`(1) man page:

- `-xia`
- `-library=[no%]interval`

Note – The C++ interval arithmetic library is compatible with interval arithmetic as implemented in the Fortran compiler.

Mixed-Language Linking

The new `-xlang` option allows you to link Fortran and C++ object files. For example, you can link a C++ main program with Fortran object files.

To determine which driver to use for mixed-language linking, use the following language hierarchy:

1. C++
2. Fortran 95 (or Fortran 90)
3. Fortran 77

When linking Fortran 95, Fortran 77, and C++ object files together, use the driver of the highest language. For example, use the following C++ compiler command to link C++ and Fortran 95 object files.

```
example% CC -xlang=f95 ...
```

To link Fortran 95 and Fortran 77 object files, use the Fortran 95 driver, as follows.

```
example% f95 -xlang=f77 ...
```

For more information, see the entries for the following in the `CC`(1) man page:

- `-xlang=f77,f90,f95`
- `-staticlib=[no%]f77,[no%]f90,[no%]f95,[no%]sunperf`

Enabling and Disabling Trigraph Recognition

The `-xtrigraphs` option determines whether the compiler recognizes trigraph sequences as defined by the ISO/ANSI C standard.

By default, the compiler assumes `-xtrigraphs=yes` and recognizes all trigraph sequences throughout the compilation unit.

If your source code has a literal string containing question marks (?) that the compiler is interpreting as a trigraph sequence, you can use the `-xtrigraph=no` suboption to turn off the recognition of trigraph sequences. The `-xtrigraphs=no` option turns off recognition of all trigraphs throughout the entire compilation unit.

Consider the following example source file named `trigraphs_demo.cc`.

```
#include <stdio.h>

int main ()
{

  (void) printf("(\?\?) in a string appears as (??)\n");

  return 0;
}
```

Here is the output if you compile this code with `-xtrigraphs=yes`.

```
example% CC -xtrigraphs=yes trigraphs_demo.cc
example% a.out
(??) in a string appears as []
```

Here is the output if you compile this code with `-xtrigraphs=no`.

```
example% CC -xtrigraphs=no trigraphs_demo.cc
example% a.out
(??) in a string appears as (??)
```

For more information on using the `-xtrigraphs` option, see the `CC`(1) man page. For information on trigraphs, see the *C User's Guide* chapter about transitioning to ANSI/ISO C.

Filtering Linker Error Messages

The new `-filt` option controls the filtering that `CC` normally applies to linker error messages.

- `-filt=no%names` suppresses the demangling of C++ mangled linker names.
- `-filt=no%returns` suppresses the demangling of return types of functions. This suppression helps you identify function names more quickly, but note that in the case of co-variant returns, some functions differ only in the return type.
- `-filt=no%errors` suppress the C++ explanation of linker error messages. This suppression is useful when the linker diagnostics are provided directly to another tool.

The following example shows the effects of using `-filt` when compiling this code.

```
// filt_demo.cc
class type {
public:
    virtual ~type(); // no definition provided
};

int main()
{
    type t;
}
```

When you compile the code without the `-filt` option, the compiler assumes `-filt=names,returns,errors` and displays the standard output.

```
example% CC filt_demo.cc
Undefined             first referenced
 symbol                  in file
type::~type()         filt_demo.o
type::__vtbl          filt_demo.o
[Hint: try checking whether the first non-inlined, non-pure
virtual function of class type is defined]

ld: fatal: Symbol referencing errors. No output written to a.out
```

The following command suppresses the demangling of the of the C++ mangled linker names and suppresses the C++ explanations of linker errors.

```
example% CC -filt=no%names,no%errors filt_demo.cc
Undefined                       first referenced
 symbol                             in file
__1cEtype2T6M_v_                    filt_demo.o
__1cEtypeG__vtbl_                   filt_demo.o
ld: fatal: Symbol referencing errors. No output written to a.out
```

For more information, see the `CC`(1) man page.

Shared `libCstd`

Sun WorkShop 6 update 1 Compilers C++ (5.2) includes a shared version of the `libCstd` library.

To use the shared version of `libCstd`, compile the program in the usual way, but use a separate link step. In the link step, use the `-library=no%Cstd` option, and put the shared library name explicitly on the command line, as shown in this example.

```
example% CC -library=no%Cstd *.o \
-o myprog /opt/SUNWspro/WS6U1/lib/libCstd.so.1
```

If Sun WorkShop 6 update 1 is not installed in `/opt`, ask your system administrator for the equivalent path.

Note – If you use `-library=no%Cstd` in a command that compiles any C++ source code, the compiler will not find C++ standard headers.

Shared `libiostream`

Sun WorkShop 6 update 1 Compilers C++ (5.2) includes a shared version of the classic iostream library, `libiostream`.

To use the shared version of `libiostream`, compile the program in the usual way, but use a separate link step. In the link step, put the shared library name explicitly on the command line, and do not use the `-library=iostream` option, as shown in this example.

```
example% CC *.o -o myprog \
/opt/SUNWspro/WS6U1/lib/lib/libiostream.so.1
```

If Sun WorkShop 6 update 1 is not installed in `/opt`, ask your system administrator for the equivalent path.

Note – You must use `-library=iostream` on each compilation of the program build, but you must not use `-library=iostream` on the link step.

Optimization Pragmas

To help the optimizer generate better code, you can use the following new pragmas:

- `#pragma no_side_effect (`*name* `...)`

  Use this pragma to indicate that the function does not change any persistent state.

- `#pragma returns_new_memory (`*name*`, ...)`

  Use this pragma when the function returns the address of the newly allocated memory and you can guarantee that the pointer does not alias with any other pointer.

For both pragmas, *name* refers to the most recently declared function that uses that name.

Place these pragmas immediately after the functions to which they refer. For example, the first code example shows the correct placement of the `no_side_effects` pragma and the second example shows an incorrect placement.

```
class good_example {
  void no_op();
  #pragma no_side_effects (no_op) // correct placement
}

class bad_example{
  void no_op();
}
#pragma no_side_effects (no_op) // incorrect placement
```

Recognition of `.c++` Extension

When a file name appears on the command line, the compiler looks at the suffix to determine how to process the file. For example, the compiler processes files ending with `.o` as object files. The Sun WorkShop 6 update 1 Compilers C++ (5.2) recognizes files with the following extensions as C++ source files.

- `.c`
- `.C`
- `.cc`
- `.cpp`
- `.cxx`
- `.c++`
- `.i`

Prefetch Latency Specifier

(*SPARC* platform) If you are running computationally intensive codes on large multiprocessors, you might find it advantageous to use the new `-xprefetch` suboption `latx:`*factor*. This suboption instructs the code generator to adjust the default latency time between a prefetch and its associated load or store by the specified factor.

For more information, see the `CC`(1) man page.

Fortran Compilers

Both the Fortran 95 and Fortran 77 compilers are released with Sun WorkShop 6 update 1.

TABLE 2-3 lists the new features available with the Sun WorkShop 6 update 1 release that are common to both the Fortran 95 and Fortran 77 compilers. Some of the features are described in greater detail in the sections following the table.

TABLE 2-3 Fortran Compilers New Features

Feature	Options	Description
Support for the UltraSPARC III Processor	`-xtarget` `-xchip`	The `-xtarget` and `-xchip` options now accept `ultra3`. See the following discussion for the recommended flags to use for optimal UltraSPARC III performance.
Support for `int2` Intrinsic		The `int2` intrinsic supports compatibility with older Fortran 77 programs. It is equivalent to the preferred Fortran 95 `int(`*var*`,2)` intrinsic for conversion of data argument *var* to a 2-byte integer.

TABLE 2-3 Fortran Compilers New Features *(Continued)*

Feature	Options	Description
Enhanced `-fast` Option	`-fast`	The `-fast` option now includes the `-xprefetch` option.
Prefetch Latency Specifier	`-xprefetch=...,latx:`*factor*	This new suboption determines how far apart the instruction the compiler generates to prefetch data will be from the subsequent use of the data in a load or store instruction.
Mixed-Language Linking	`-xlang`	The new `-xlang` option enables linking of mixed Fortran and C++ object files.

Support for the UltraSPARC III Processor

Use the following options to compile for optimal performance when compiling and running your program on an UltraSPARC III processor:

```
-fast -xcrossfile -xprofile={collect:|use:}
```

- For cross-compilation (compiling on a platform other than UltraSPARC III but generating object binaries to run on an UltraSPARC III system), add the following options to insure correct cache sizes and optimization strategies:

  ```
  -xtarget=ultra3 -xarch={v8plusb|v9b}
  ```

- Specify `-xarch=v8plusb` for 32-bit code generation and `v9b` for 64-bit code. For programs accessing very large data files, 64-bit code provides better performance. However, `-xarch=v9b` should only be used when 64-bit code generation is required. In some cases it can result in slower performance.
- The `-fast` option enables optimizations that favor speed of execution and raises the optimization level to `-xO5`. Note that `-fast` allows substitution of faster arithmetic operations where possible (`-fsimple=2`), so some inaccuracies might result.
- `-xcrossfile` enables the compiler to apply optimizations across all the source files specified on the command line, including some interprocedural optimizations.
- `-xprofile={collect:|use:}` enables program performance profiling. Profiling allows the compiler to identify the most frequently executed sections of code and to perform localized optimizations to best advantage.

Note – Programs compiled specifically for the UltraSPARC III platform with `-xarch={v8plusb|v9b}` will not operate on platforms other than the UltraSPARC III platform. Use `-xarch={v8plusa|v9a}` to compile programs to run compatibly on UltraSPARC I, UltraSPARC II, and UltraSPARC III platforms.

Support for `int2` Intrinsic

The Fortran 95 and Fortran 77 compilers now support the `int2` intrinsic for conversion of data types to 2-byte integer. Use of `int2` as an intrinsic appears in many legacy Fortran 77 codes in the form `M=int2(J)`.

The preferred Fortran 95 usage is `M=int(J,2)`.

Enhanced `-fast` Option

The `-xprefetch` option has been added to the list of options included in the `-fast` option. This enables the compiler to strategically generate prefetch instructions. Using the `-xprefetch` option can add a substantial performance gain in code with loops that process data. The prefetch mechanism of the UltraSPARC III platform is much improved over that used by the UltraSPARC II platform.

Prefetch Latency Specifier

(*SPARC* platform) If you are running computationally intensive codes on large multiprocessors, you might find it advantageous to use the new `-xprefetch` suboption `latx:`*factor*. This suboption instructs the code generator to adjust the default latency time between a prefetch and its associated load or store by the specified factor.

See the `f77`(1) and `f95`(1) man pages for details.

Mixed-Language Linking

The new `-xlang` option allows you to link Fortran and C++ object files. For example, you can link a Fortran main program with C++ object files.

To determine which driver to use for mixed-language linking, use the following language hierarchy:

1. C++
2. Fortran 95
3. Fortran 77

When linking Fortran 95, Fortran 77, and C++ object files together, use the driver of the highest language. For example, use the following C++ compiler command to link C++ and Fortran 95 object files.

```
example% CC -xlang=f95 ...
```

To link Fortran 95 and Fortran 77 object files, use the Fortran 95 driver, as follows.

```
example% f95 -xlang=f77 ...
```

Interval Arithmetic

TABLE 2-4 lists the new features available with the Sun WorkShop 6 update 1 release of interval arithmetic.

TABLE 2-4 Interval Arithmetic New Features

Feature	Description
Interval Arithmetic Support for C++	(*SPARC* platform) This release of the C++ compiler provides a C++ interface to the interval arithmetic library.
New `f95` `INTERVAL` Intrinsic Operators and Functions	`f95` interval arithmetic adds support for the dependent subtraction operator, a division with intersection function, and the interval version of the generic random number generation subroutine `RANDOM_NUMBER`.

Interval Arithmetic Support for C++

The Sun WorkShop 6 update 1 release includes a C++ version of the interval functions and operators that are contained in Fortran 95 Interval Arithmetic. For more information about the C++ interval arithmetic library, see the *C++ Interval Arithmetic Programming Reference* available from `http://docs.sun.com` (in the Forte™ Developer 6 update 1/Sun WorkShop 6 update 1 Collection).

Interval Arithmetic support for C++ provides a C++ header file and library that implements three interval classes, one each for float, double, and long double. The interval classes include:

- Interval arithmetic operations and mathematical functions that form a closed mathematical system, which means that valid results are produced for any possible operator-operand combination, including division by zero and other indeterminate forms involving zero and infinities
- Three types of interval relational functions:
  - Certainly
  - Possibly
  - Set
- Interval-specific functions, such as `intersect` and `interval_hull`
- Interval-specific functions, such as `inf`, `sup`, and `wid`
- Interval input/output, including single-number input/output

For standard mode compilation, all symbols in the library are in the namespace `SUNW_interval`.

The compilation interface consists of the following:

- A new value, `interval`, for the `-library` flag, which expands to the appropriate libraries.
- A new value, `interval`, for the `-staticlib` flag, which at present is ignored because only static libraries are provided with this release.
- A new flag, `-xia`, which is the same flag used with the Fortran compilers, although the expansion is different.

To use the C++ interval arithmetic features, add the following header file to the code.

```
#include <suninterval.h>
```

An example of compiling code using the `-xia` command-line option is shown below.

```
example% CC -o filename -xia filename.cc
```

New `f95 INTERVAL` Intrinsic Operators and Functions

The following features have been added to `f95` interval arithmetic:

- Dependent Subtraction Operator
- Division With Intersection Function
- Random Number Subroutine

Dependent Subtraction Operator

The dependent subtraction operator `.DSUB.` can be used to recover either operand of an interval arithmetic addition.

Two interval variables are dependent when one interval variable is a result of an interval arithmetic operation applied to the other interval variable. For example, if X = A + B, then X depends on both A and B. Dependent interval subtraction produces narrower interval results when recovering A or B from X.

Note – Dependent operations cannot be applied to interval constants because constants are independent. Applying dependent operations to interval constants produces a compile-time error.

The result of `X.DSUB.A` contains the solution for B of the interval equation X = A + B. The result is `[-inf,inf]` if there is no solution.

- **Arguments.** `X` and `A` must be intervals with the same kind type parameter value.
- **Result type.** Same as `X`.

The following examples show the behavior of `DSUB.`

- `X .DSUB. X = [0]` if $-\infty < \underline{x} \leq \bar{x} < +\infty$
- `X .DSUB. A = [-inf,inf]` if `WID(X)` < `WID(A)` because `WID(X)` must be greater than or equal to `WID(A)` if X = A + B.
- `[empty] .DSUB. [empty] = [-inf,inf]`
- `[empty] .DSUB. A = [empty]` if $A \neq$ `[empty]`

- `X .DSUB. [empty] = [-inf,inf]` if `X` ≠ `[empty]`. This result follows because `X` ≠ `[empty]` and `A` ≠ `[empty]` is an impossible combination.

Division With Intersection Function

The function `DIVIX` returns C ∩ (B/A), the interval enclosure of the result of the interval division operation (B/A) intersected with the interval C.

In the case when A contains zero, the mathematical result of the interval division operation (B/A) is the union of two disjoint intervals. Each interval in the union can be represented in the currently implemented interval arithmetic system. The `DIVIX` function allows one or part of these intervals to be returned.

- **Arguments.** `A`, `B`, and `C` must be intervals with the same kind type parameter value.
- **Result type.** Same as `A`.

The following example shows the output of `DIVIX`.

```
DIVIX( [3.0,5.0] , [6.0,15.0] , [2.0,6.0] )== [2.0,5.0]
DIVIX( [-3.0,5.0] , [8.0,20.0] , [-0.5,1.0] )== [EMPTY]
DIVIX( [-3.0,5.0] , [8.0,20.0] , [-7.0,8.0] )== [-7.0,8.0]
DIVIX ([-1,1], [1], [-Inf,0.0E+0]) == [-Inf,-1.0]
DIVIX ([-1,1], [1], [0,inf])       == [1.0,Inf]
```

Random Number Subroutine

`RANDOM_NUMBER(HARVEST)` returns through the interval variable `HARVEST` one pseudorandom interval or an array of pseudorandom intervals $[\underline{x}, \bar{x}]$.

$0 \le \underline{x} \le \bar{x} < 1$ holds for the interval endpoints, where $\underline{x}$ is the lower endpoint, and $\bar{x}$ is the upper endpoint. Therefore, $\underline{x}$ is uniformly distributed on the interval [0, 1], and, given $\underline{x}$, $\bar{x}$ is uniformly distributed on the interval $[\underline{x}, 1]$.

dbx

TABLE 2-5 lists the new features available with the Sun WorkShop 6 update 1 release of `dbx`.

TABLE 2-5 `dbx` New Features

Feature	Description
Support for Interval Arithmetic Expressions in Fortran	Fortran interval types and expressions are now supported. Simple arithmetic (add, subtract, multiply, divide, negate), equal, and not equal operations are implemented.

Sun Performance Library

Sun Performance Library™ is a set of optimized, high-speed mathematical subroutines for solving linear algebra and other numerically intensive problems. Sun Performance Library is based on a collection of public domain applications available from Netlib at `http://www.netlib.org`. These routines have been enhanced and bundled as the Sun Performance Library.

TABLE 2-6 lists the new features and enhancements available with the Sun WorkShop 6 update 1 release of the Sun Performance Library.

TABLE 2-6 Sun Performance Library New Features

Feature	Description
Additional Performance Enhancements for the UltraSPARC III Processor	Optimized performance on both single-processor and multi-processor UltraSPARC III platforms.
Fast Fourier Transform (FFT) Improvements	Made changes to man pages, added additional error-checking to the code, and made performance improvements.
FFT Documentation	Created *Using Sun Performance Library Fast Fourier Transform Routines* document that describes `FFTPACK` and `VFFTPACK` routines. *Using Sun Performance Library Fast Fourier Transform Routines* is available from `http://docs.sun.com` in the Forte™ Developer 6 update 1/Sun WorkShop 6 update 1 Collection.

TABLE 2-6 Sun Performance Library New Features *(Continued)*

Feature	Description
Sparse Solver	Improved performance and functionality, extend messaging system, incorporated SuperLU.
Improved Parallelization and Scalability	Used data flow programming techniques on selected routines and libraries to improve performance.
Open MP Directives Used	Increased support for Open MP directives.
Interval Matrix-Multiply Implemented	Developed an interval-based `MATMUL` for the `f95` intrinsics, which is the non-transpose, non-transpose case of the `GEMM` and `GEMV` routines.

Sampling Analyzer

TABLE 2-7 lists the new features available with the Sun WorkShop 6 update 1 release of the Sampling Analyzer.

TABLE 2-7 Sampling Analyzer New Features

Feature	Description
Hardware Counter Overflow Profiling	Hardware counter overflow profiling records the callstack of each light-weight process (LWP) at the time the hardware counter of the CPU on which the LWP is running overflows.
Standalone `collect` Command	The new `collect` command allows you to collect performance data on your applications independent of Sun WorkShop and `dbx`.
Improved Support for MPI Applications	The new `collect` command provides improved message-passing interface (MPI) support.
Improved Support for OpenMP (`libmtsk`) Applications	You can now distinguish when a slave thread is waiting for synchronization at the end of a parallel region and when it is waiting because the code is in a serial region.
Improved Mapfile Generation	The mapfile is now produced so that it orders the executable by whatever metric is being used for sorting the function list.
Additions to the Select Filters Dialog Box	The Experiment list and Select All, Clear All, and Reverse buttons in the Select Filters dialog box let you select experiments for which you want to change the data displayed. The Enable All, Enable All Selected, Disable All, and Disable All Selected buttons let you enable and disable data display for experiments.

Hardware Counter Overflow Profiling

Hardware counter overflow profiling records the callstack of each light-weight process (LWP) at the time the hardware counter of the CPU on which the LWP is running overflows. The data recorded includes a timestamp and the IDs of the CPU, the thread, and the LWP. Hardware counter overflow profiling can be done only on UltraSPARC III systems running the Solaris 8 Operating Environment *SPARC Platform Edition* and on Intel systems (Pentium II or III) running the Solaris 8 Operating Environment *Intel Platform Edition*. For more information, see "Hardware Counter Overflow Profiling Data" and "Choosing the Data to Collect" in the Using the Debugging Window section of the Sun WorkShop online help.

Standalone `collect` Command

The new `collect` command allows you to collect performance data on your applications independent of Sun WorkShop and `dbx`. It provides arguments for specifying the types of data to be collected, naming experiments and experiment groups, and requesting that the target process be left stopped on the exit from the `exec` system call, in order to allow a debugger to attach to it. For more information, see the `collect`(1) man page.

Improved Support for MPI Applications

The new `collect` command provides the following improved message-passing interface (MPI) support:

- The `collect` command lets you specify experiment groups, which allows experiments from all the processes of an MPI run to be grouped and processed together.
- Synchronization delay tracing records all calls to the various thread synchronization routines where the real-time delay in the call exceeds a specified threshold.

CHAPTER 3

Sun WorkShop 6 New Features

This chapter describes the new features of the Sun WorkShop™ 6 compilers and tools. Tables summarize new features, and explanations of some of the new features follow the summary tables.

This chapter has the following sections:

Key Features

The following key features are the highlights of this release:

- Additional ANSI/ISO C++ compliance
- Fortran 95 compiler with OpenMP parallelization directives
- Support for the UltraSPARC™ III instruction set architecture
- Easier-to-use programming environment
- New Performance Analysis tool
- Fortran 95 interval arithmetic
- Installation improvements
- Manuals, man pages, readmes, and online help in HTML

C Compiler

TABLE 3-1 lists the new features available with the release of the Sun WorkShop 6 C compiler. These features enhance the capabilities of the C compiler and `lint` source-code checker.

TABLE 3-1 C Compiler New Features

Feature	Description
`__func__`	The C compiler predefines a static, constant, char array named`__func__` for every function definition. The array is initialized with the name of the function and can be used anywhere a static function scope array can be used, such as when printing the name of the enclosing function.
Variable argument macro	The C preprocessor accepts a variable number of arguments for a `#define` macro. If the macro definition includes an ellipsis as part of the identifier list, then there will be more arguments when the macro is invoked than there were parameters in the macro definition.
`SUNW_MP_THR_IDLE`	You can use the `SUNW_MP_THR_IDLE` environment variable to control whether a thread continues to use system resources after it finishes its task or it "sleeps."
large arrays	The C compiler supports significantly larger array objects. For specifics, see Appendix A of the *C User's Guide*.
`-errchk=locfmtchk`	`lint` accepts a new flag, `-errchk=locfmtchk`, which checks for `printf`-like format strings during the first pass of `lint`.
new `lint` directive (`PRINTFLIKE(`*n*`)`)	`lint` accepts a new directive that identifies all calls to the `printf()` family through a pointer. All such calls through the pointer can be checked for argument consistency by `lint`.
`-errwarn=t`	The C compiler and the `lint` source code checker support a new option, `-errwarn=t`, which causes the compiler to exit with a failure status if any of the specified warnings are issued.
`-errchk`	The `-errchk` option of `lint` has a new value `signext`, which you can use in conjunction with the `longptr64` option: `-errchk=longptr64,signext`. This new option warns about sign extension in order to facilitate migration to the 64-bit development environment.

TABLE 3-1 C Compiler New Features *(Continued)*

Feature	Description
`-xchar_byte_order`	The `-xchar_byte_order` option produces an integer constant by placing the characters of a multi-character character-constant in the specified byte order.
`-xinline`	The `-xinline` option accepts two new values: `%auto` and `no%`*function_name*. The `%auto` value takes effect at the `-x04` level of optimization and allows the compiler to automatically inline functions in addition to the listed function for `-xinline`. The `%no` *function_name* value tells the compiler to not inline the function *function_name*.
`-xmemalign`	The C compiler offers a new option named `-xmemalign`. This option controls code generated for potentially misaligned memory accesses and controls program behavior in the event of a misaligned access.
`-xprefetch`	You can use the `sun_prefetch.h` header file and the `-xprefetch` option to specify explicit prefetch instructions.
`-xvector`	The `-xvector` option enables automatic generation of calls to the vector library functions.

C++ Compiler

TABLE 3-2 lists the new features available with the release of the Sun WorkShop 6 C++ compiler. Some of these features are described more completely in the sections following the table.

TABLE 3-2 C++ Compiler New Features

Feature	Description
Partial Specialization	A template can be partially specialized, meaning that only some of the template parameters are specified or that one or more parameters are limited to certain categories of type.
Explicit Function Template Argument	If a template argument cannot be deduced from the function arguments, you can specify it explicitly using the syntax `f<`*template args*`>(`*function args*`)`.

Non-Type Function Template Parameters	This release supports non-type function template parameters, such as:
	`template<int I> void foo( int a[I] ) { ... }`
	`template<int I> void foo( mytype<I> m ) { ... }`

TABLE 3-2 C++ Compiler New Features *(Continued)*

Feature	Description
Member Templates	In standard mode, classes and class templates can have templates as members.
Definitions-Separate Template Organization Restriction Removed	The compiler no longer has a restriction against "definitions-separate template organization" for `-instances != extern` (that is, `-instances=explicit`, `-instances=global`, `-instances=semiexplicit`, or `-instances=static`). Regardless of the `-instances` setting, by default the compiler now includes separate source files in the search for definitions.
Prefetch Instructions	You can use `-xprefetch` in conjunction with the header file `<sun_prefetch.h>` to specify prefetch instructions on those architectures that support prefetch, such as the UltraSPARC II instruction set architecture (`-xarch=v8plus`, `v8plusa`, `v8plusb`, `v9`, `v9a`, or `v9b`).
Extern Inline Functions	This version of the compiler allows extern inline functions. If there is any local static data in an inline function, only one copy of the static data is used in all compilation units. However, the addresses of inline functions taken in different translation units will not compare as equal.
Ordering of Static Variable Destruction	The standard has defined the order of destruction of objects with static storage duration more fully; static objects must be destroyed in the reverse order of their construction. Previous language definitions left some aspects unspecified.
Sub-Aggregate Initialization	When using brace-initialization of class objects (for types where brace-initialization is allowed), the C++ standard permits a member which is itself an aggregate class to be initialized by a value of its own type.
Using Your Own C++ Standard Library	By specifying the `-library=no%Cstd` option, you can use your own version of the C++ standard library, instead of the version supplied with the compiler.
Cache Versioning	The C++ compiler has the ability to detect cache version differences and issue the appropriate error message.
Restrictions on Bitfield Size Removed	The restriction on the size of a bitfield to 32 or less is removed. Bitfields can be any size.

TABLE 3-2 C++ Compiler New Features *(Continued)*

Feature	Description
Warnings About Conversions Between Pointer-To-Function and `void*`	Previously, the compiler issued warnings about conversions between pointer-to-function and `void*`. The compiler only issues these warnings when you use the `+w2` option.

New and Changed Options	The following list shows the new and changed options.
	• New Options:
	`-xcrossfile`
	`-Bsymbolic`
	`-features=[no%]strictdestrorder`
	`-template=extdef`.
	• Changed Options:
	`-fast`
	`-library=[no%]Cstd`, `+p`
	`-ptr`, `-xprefetch`
	For more information, see the entry for each option in the *C++ User's Guide* (accessible from the `http://docs.sun.com` Web site).

Partial Specialization

A template can be fully specialized, meaning that an implementation is defined for specific template arguments. See the following code example.

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

A template can also be partially specialized, meaning that only some of the template parameters are specified, or that one or more parameters are limited to certain categories of type. The resulting partial specialization is itself still a template. The following examples use the previous primary template.

- Special template definition for cases when the first template parameter is type `int`. See the following code example.

```
template<classU> class A<int> { ... };
```

- Special template definition for cases when the first template parameter is any pointer type. See the following code example.

```
template<class T, class U> class A<T*> { ... };
```

- Special template definition for cases when the first template parameter is pointer-to-pointer of any type, and the second template parameter is type `char`. See the following code example.

```
template<class T> class A<T**, char> { ... };
```

Explicit Function Template Argument

If a template argument cannot be determined from the function arguments, you can now explicitly specify the template argument using the syntax `f<`*template args*`>(`*function args*`)`. See the following code example.

```
template<class Mytype> Mytype* construct(float, float);
...
int* x = construct<int>(a, b);
```

Non-Type Function Template Parameters

This release supports non-type function template parameters, as illustrated in the following code example.

```
template<int I> void foo( int a[I] ) { ... }
template<int I> void foo( mytype<I> m ) { ... }
```

This release does not allow expressions involving non-type template parameters in the function parameter list, as illustrated in the following code example.

```
// these are not supported
template<int I> void foo( mytype<2*I> ) { ... }
template<int I, int J> void foo( int a[I+J] ) { ... }
```

Member Templates

In standard mode, classes and class templates can have templates as members, as illustrated in the following code example.

```
template <class T1>
class OuterClass {
public:
    // class member template
    template <class T2>
        class MemberClass
    {
        T2 MCmember;
        T1 OCmember;
    };
    template<class T3> operator T3() { ... }
...
};
```

Note – Member templates are not supported in compatibility mode (`-compat[=4]`).

Definitions-Separate Template Organization Restriction Removed

The compiler no longer has a restriction against "definitions-separate template organization" for `-instances != extern` (that is, `-instances=explicit`, `-instances=global`, `-instances=semiexplicit` or `-instances=static`). Regardless of the `-instances` setting, the compiler will now, by default, include separate source files in the search for definitions.

To turn this restriction back on, use the `-template=no%extdef` option. Note, however, that when the `-template=no%extdef` option is specified, the compiler does not search for separate source files even with `-instances=extern`.

Ordering of Static Variable Destruction

The standard has defined the order of destruction of objects with static storage duration more fully; static objects must be destroyed in the reverse order of their construction. Previous language definitions left some aspects unspecified.

This stricter ordering is implemented for standard mode only. In compatibility mode (`-compat[=4]`), the order of destruction is implemented as before.

If your program depends on a particular order of destruction and worked with an older compiler, the order required by the standard might break the program in standard mode. The `-features=no%strictdestrorder` command option disables the strict ordering of destruction.

Sub-Aggregate Initialization

When using brace-initialization of class objects (for types where brace-initialization is allowed), the C++ standard permits a member that is itself an aggregate class to be initialized by a value of its own type. See the following code example.

```
struct S {  // an aggregate type
int i, j;
};
struct T { // an aggregate type
S s; // aggregate member
int k;
};
T t1 = { {1, 2}, 3 }; // traditional initialization
S s1 = { 1, 2 };
T t2 = { s1, 3 }; // sub-aggregate initialization
```

Using Your Own C++ Standard Library

If you want to use your own version of the C++ standard library instead of the version supplied with the compiler, you can do so by specifying the `-library=no%Cstd` option. This option prevents finding any of the following headers:

`<algorithm>` `<bitset>` `<complex>` `<deque>` `<fstream>` `<functional>` `<iomanip>` `<ios>` `<iosfwd>` `<iostream>` `<istream>` `<iterator>` `<limits>` `<list>` `<locale>` `<map>` `<memory>` `<numeric>` `<ostream>` `<queue>` `<set>` `<sstream>` `<stack>` `<stdexcept>` `<streambuf>` `<string>` `<strstream>` `<utility>` `<valarray>` `<vector>`

When `-library=no%Cstd` is specified, the `libCstd` library, which implements those headers, is not automatically linked with your program. To use any of the features declared in the above headers, you must use the `-I` option to point to the directory where the replacement headers are located, and you must link your program with a library or set of object files containing the implementation of the replacement headers.

You cannot reliably replace only a portion of the headers listed above, nor can you reliably link `libCstd` with all or part of another library implementation. For example, you cannot replace only the string classes and use `libCstd` for everything else. Either use the library supplied with the compiler, or replace all of the functionality listed above.

The remaining headers (`<exception>`, `<new>`, `<typeinfo>`, and all the headers inherited from C) are integral to the compiler itself or to Solaris, and are not affected by the `-library=no%Cstd` option. Linking of the library `libCrun` also is not affected by the `-library=no%Cstd` option.

There is no mechanism to replace any of the functionality of `libCrun`. If you replace the standard library, the code must be compiled with the versions of `<exception>`, `<new>`, and `<typeinfo>` supplied with the compiler. In standard mode (the default mode) C++ programs must be linked with `libCrun`.

Note – This option is available to "use at your own risk." Using your own version of the C++ standard library might not produce optimal results.

Cache Versioning

The C++ compiler now has the ability to detect cache version differences and issue the appropriate error message. The compiler marks each template cache directory with a version string that uniquely identifies the template cache version. Subsequent releases of the compiler will also use cache version strings, although these versions may be different from the current version.

This compiler and future compilers will detect the version strings from within the cache directories and issue an error as appropriate. For example, a future compiler that uses a different template cache version string and processes a cache directory produced by this release of the compiler might issue the following error.

```
SunWS_cache: Error: Database version mismatch
/SunWS_cache/CC_version
```

Similarly, this release of the compiler issues an error if it encounters a cache directory produced by a future compiler.

The template cache directories produced by the Sun WorkShop C++ compiler 5.0 compiler are not versioned. However, the Sun WorkShop 6 C++ compiler processes these cache directories without an error or a warning. These cache directories are converted to the cache directory format used by the Sun WorkShop 6 C++ compiler.

A template cache directory produced by the Sun WorkShop 6 C++ compiler or later releases cannot be used by the Sun WorkShop C++ compiler 5.0. The Sun WorkShop C++ compiler 5.0 is not capable of recognizing format differences and it will issue an assertion.

Fortran Compilers

Sun WorkShop 6 includes the Sun WorkShop™ Compilers Fortran 77 and Sun WorkShop™ Compilers Fortran 95.

The Fortran compilers in this Sun WorkShop 6 release support only versions 2.6, 7, and 8 of the Solaris *SPARC™ Platform Edition* Operating Environment. For the Solaris Intel IA-32 platform, Sun has discontinued development of Fortran compilers and the Sun Performance Library. Sun does not offer Forte™ for High Performance Computing or the Forte™ Fortran Desktop Edition (formerly known as Sun Performance WorkShop Fortran) for the Solaris Intel IA-32 platform in this release. You can contact The Portland Group (`http://www.pgroup.com`) for information about their line of software development tools for the Solaris IA-32 platform.

Fortran 77 Compiler

TABLE 3-3 lists the new features available with the release of the Sun WorkShop 6 Fortran 77 compiler. Some of these features are described more completely in "New Fortran Compiler Features" on page 52.

TABLE 3-3 Fortran 77 Compiler New Features

Feature	Description	
Effect of `FORM="BINARY"` on I/O Operations	Specifying this new option in an `OPEN(...)` statement causes the file to be treated as a sequential binary (unformatted) file with no record marks. This enables data to be written and read as a continuous stream of bytes, and provides compatibility with other vendor systems. It is implemented in both the Fortran 95 and Fortran 77 compilers.	
Debugging Optimized Code	The restrictions on compiling with `-g` have been relaxed so that it is now possible to compile at `-O4` and `-O5` or any of the parallelization flags (`-parallel`, `-explicitpar`, `-autopar`) with debugging (`-g`).	
New Command-Line Flags	The following new command-line flags appear in this release of `f77` (see the `f77`(1) man page):	
	`-aligncommon`	Aligns common block elements to specified byte boundaries
	`-r8const`	Promotes single-precision data constants to `REAL*8`
	`-xmemalign`	Specifies general alignment of data elements
Expanded Command-Line Flags	The following `f77` command-line flags have been expanded (see the `f77`(1) man page):	
	`-fast`	Sets `-O5, -fsimple=2, -xvector=yes, -pad=common`
	`-xprefetch`	Enables explicit pragma prefetch directives to force generation of prefetch instructions on UltraSPARC platforms
	`-xtypemap`	Includes an expanded set of possible data type specifications
Cray-Style Directives	`AUTOSCOPE` qualifier added to Cray-style parallelization directives.	
Licensing	The parallelization features of the Fortran 77 compiler require a Sun WorkShop HPC license.	
Hyper-Linked Compiler Diagnostics	`f77` error messages in the Sun WorkShop Building window now have hyperlinks to help pages that explain the messages.	

Fortran 95 Compiler

TABLE 3-4 lists the new features available with the release of the Sun WorkShop 6 Fortran 95 compiler. Some of these features are described more completely in "New Fortran Compiler Features" on page 52.

TABLE 3-4 Fortran 95 Compiler New Features

Feature	Description	
Compliance	The Fortran 95 compiler is fully compliant with the Fortran 95 standard.	
New Command	The Fortran 95 compiler is invoked by both the `f90` and `f95` command. The `f95` command is new. `f90` is equivalent to `f95`.	
File Extensions	The compiler will accept free-format source files with `.f95` and `.F95` extensions as well as `.f90` and `.F90.`	
Effect of `FORM="BINARY"` on I/O Operations	Specifying this new option in an `OPEN(...)` statement causes the file to be treated as a sequential binary (unformatted) file with no record marks. The enables data to be written and read as a continuous stream of bytes, and provides compatibility with other vendor systems. It is implemented in both the Fortran 95 and Fortran 77 compilers. See the *FORTRAN 77 Language Reference.*	
Debugging Optimized Code	The restrictions on compiling with `-g` have been relaxed so that it is now possible to compile at `-O4` and `-O5` or any of the parallelization flags (`-parallel`, `-explicitpar`, `-autopar`) with debugging (`-g`).	
`f77` Flags	Most of the `f77` compiler flags are now implemented in `f95`. See the `f95`(1) man page for details. These include:	
	`-erroff`	Turns off selected error messages
	`-errtags`	Displays error messages with tags
	`-ext_names`	Creates external names with or without underscores
	`-fpp`	Specifies source code preprocessor
	`-loopinfo`	Shows which loops are parallelized
	`-sbfast`	Produces browser table information
	-silent	Suppresses compiler messages
	-U	Allows lowercase and uppercase
	-u	Implies `IMPLICIT NONE`
	-xcrossfile	Enables optimization across files
	-xF	Allows function-level reordering for Analyzer
	-xinline	Compiles functions inline
	`-xtypemap`	Specified default data sizes

TABLE 3-4 Fortran 95 Compiler New Features *(Continued)*

Feature	Description	
New Flags	The following new flags are implemented in `f95`:	
	`-aligncommon`	Aligns common block elements to specified byte boundaries
	`-mp=openmp`	Accepts OpenMP directives
	`-r8const`	Promotes single-precision constants to `REAL*8`
	`-xia`	Enables processing of `INTERVAL` data types (recommended)
	`-xinterval`	Enables processing of `INTERVAL` data types
	`-xmemalign`	Specifies general alignment of data elements
	`-xrecursive`	Allows recursive calls without `RECURSIVE` attribute
Expanded Flags	The following `f95` command-line flags have been expanded (see the `f95`(1) man page):	
	`-fast`	Sets `-O5 -fsimple=2 -xvector=yes -pad=common`
	`-xprefetch`	Enables explicit pragma prefetch directives to force generation of prefetch instructions on UltraSPARC platforms
	`-xtypemap`	Includes an expanded set of possible data type specifications
OpenMP	This release of Fortran 95 implements the OpenMP interface for explicit parallelization, including a set of source code directives, runtime library routines, and environment variables. See the *Fortran User's Guide.*	
Cray-Style Directives	`AUTOSCOPE` qualifier added to Cray-style parallelization directives.	
Interval Arithmetic Extensions	This release of Fortran 95 implements extensions to support intrinsic `INTERVAL` data types.	
Licensing	The parallelization features of the Fortran 95 compiler require a Sun WorkShop HPC license.	
Hyper-Linked Compiler Diagnostics	Sun WorkShop Building window now interprets `f95` error messages as live links into the online help.	

New Fortran Compiler Features

The following sections describe some of the new Fortran compiler features in greater detail.

Effect of `FORM="BINARY"` on I/O Operations

- `WRITE` statement—Data is written to the file in binary, with as many bytes transferred as there is specified in the output list.
- `READ` statement—Data is read into the variables on the input list, with as many bytes transferred as demanded by the list. Because there are no record marks on the file, there will be no "end-of-record" error detection. The only error detected is end-of-file, or abnormal system errors.
- `INQUIRE` statement—`INQUIRE` on a file opened with `FORM="BINARY"` returns the following information:

```
FORM="BINARY"
ACCESS="SEQUENTIAL"
SEQUENTIAL="YES"
DIRECT="NO"
FORMATTED="NO"
UNFORMATTED="YES"
RECL= and NEXTREC= are undefined.
```

- `BACKSPACE` statement—Not allowed; returns an error.
- `ENDFILE` statement—Truncates file at current position, as usual.
- `REWIND` statement—Repositions file to beginning of data, as usual.

OpenMP

(*Fortran 95* only) This release of Fortran 95 implements the OpenMP interface for explicit parallelization, including a set of source code directives, runtime library routines, and environment variables. Preliminary documentation is available in the OpenMP README. The OpenMP specifications can be viewed at `http://www.openmp.org/`.

A summary of all directives accepted by the Fortran compilers, including OpenMP, can be found in Appendix E of the *Fortran User's Guide*. See also the *Fortran Programming Guide* for additional information on the parallelization features of the Fortran compilers.

Note – The parallelization features of the Fortran compilers require a Sun WorkShop HPC license.

Interval Arithmetic Extensions

(*Fortran 95* only) This release of Fortran 95 includes interval arithmetic extensions. See "Fortran 95 Interval Arithmetic" on page 53.

Hyper-Linked Compiler Diagnostics

When you use Sun WorkShop to build and compile applications, `f77` and `f95` diagnostic messages in the Building window now have hyperlinks to help pages. Clicking on the error message launches a help browser with additional information about the specific error diagnostic.

Fortran 95 Interval Arithmetic

Support for intrinsic `INTERVAL` data types is a new feature in the Sun WorkShop 6 Fortran 95 compiler.

Two new compiler flags, `-xia` and `-xinterval`, tell the compiler to recognize interval-specific languages extensions and generate the code to implement interval instructions.

What Is Interval Arithmetic?

Interval arithmetic is used to evaluate arithmetic expressions over sets of numbers contained in intervals. An interval is the set of all real numbers between and including the interval's lower and upper bound. Any interval arithmetic result is a new interval that is guaranteed to contain the set of all possible resulting values.

With Sun WorkShop 6 Fortran 95, it is a simple matter to write interval programs to compute rigorous bounds on the value of arithmetic expressions:

- Declare variables to be type `INTERVAL`.
- Write normal Fortran code using the intrinsic `INTERVAL` functions and operators, relational operators, and format edit descriptors.
- Compile the code using the `-xia` command-line option.

To achieve the best results, use existing interval algorithms that compute narrow width interval results. Devising algorithms to compute narrow interval results is the topic of interval analysis.

Why Is Interval Arithmetic Important?

Interval arithmetic is important for the following reasons:

- Interval arithmetic can be used to perform machine computations with guaranteed bounds on errors from all sources, including input data errors, machine rounding, and their interactions.
- Interval algorithms can be developed that solve nonlinear problems, such as the solution to nonlinear systems of equations and nonlinear programming.

As intervals become more widely used, libraries of interval solvers will be used routinely to compute sharp interval solutions to linear and nonlinear problems, while taking into account all sources of error. With these libraries, scientists, engineers, and developers of commercial applications will be able to write programs to solve problems that are currently out of reach.

Where Can I Get More Information?

See the *Interval Arithmetic Programming Reference* or the list of online resources in the Interval Arithmetic readme.

dbx

TABLE 3-5 lists the new features available with the release of Sun WorkShop 6 `dbx`.

TABLE 3-5 `dbx` New Features

Feature	Description
`$firedhandlers` ksh variable	The read only ksh variable `$firedhandlers` has been added. This variable can be used in conjunction with the `delete` and `handler` commands as an alternative to the clear command. For more information, see "Variables," "`delete` Command," and "`handler` Command" in the Using `dbx` Commands section of the Sun WorkShop online help.
Partial clearing of breakpoints	The `clear` command now facilitates partial clearing of In Class, In Method, and In Function breakpoints. For more information, see "`clear` Command" in the Using `dbx` Commands section of the Sun WorkShop online help.

TABLE 3-5 dbx New Features *(Continued)*

Feature	Description
Trace output	Output of traces can be redirected to files. For more information, see "`trace` Command" in the Using `dbx` Commands section of the Sun WorkShop online help.
New `dbx` environment variables	• The new `dbx` environment variable `stack_find_source` controls whether `dbx` automatically moves up the call stack to a frame with debuggable source code when the program stops. For more information, see "`stack_find_source` Environment Variable" in the Using `dbx` Commands section of the Sun WorkShop online help. • The new `dbx` environment variable `proc_exclusive_attach` controls whether `dbx` can attach to a process that is under the control of another debugger or debugging tool. For more information, see "`proc_exclusive_attach` Environment Variable" in the Using `dbx` Commands section of the Sun WorkShop online help. • The new `dbx` environment variable `step_granularity` controls whether the `step` and `next` commands work on statements or lines or source code. For more information, see "`step_granularity` Environment Variable" in the Using `dbx` Commands section of the Sun WorkShop online help. • The new `dbx` environment variable `mt_scalable` helps debugging multithreaded applications with many LWPs (lightweight processes) by reducing resource usage. For more information, see "`mt_scalable` Environment Variable" in the Using `dbx` Commands section of the Sun WorkShop online help. • The new `dbx` environment variable `rtc_error_stack` determines whether stack traces show frames corresponding to RTC internal mechanisms. For more information, see "`rtc_error_stack` Environment Variable" in the Using `dbx` Commands section of the Sun WorkShop online help. • The new `dbx` environment variable `rtc_inherit` determines whether runtime checking is enabled on child processes that are executed from the debugged program. For more information, see "`rtc_inherit` Environment Variable" in the Using `dbx` Commands section of the Sun WorkShop online help. • The new `dbx` environment variable `rtc_use_traps` enables a workaround for the eight megabyte code limitation on runtime checking. For more information, see "`rtc_use_traps` Environment Variable" in the Using `dbx` Commands section of the Sun WorkShop online help.

TABLE 3-5 `dbx` New Features *(Continued)*

Feature	Description
Breakpoints in C++ member functions	Breakpoints in C++ inlined member functions work correctly. This includes proper evaluation of handler conditions (`-if`).
LWP-related commands now always available	LWP-related commands are now always available, not just when you are debugging a multithreaded application. For more information, see "`lwp` Command" and "`lwps` Command" in the Using `dbx` Commands section of the Sun WorkShop online help.
Interrupting a running process	When Control-C does not seem to stop a hung process, the third consecutive Control-C will force the process to stop by other means. See "Interrupting a Running Process" in the Using `dbx` Commands section of the Sun WorkShop online help.
Full line number information	Full line number information is now recorded when you compile with the `-g` and `-O` options. You can now use the `step` command and `next` command to step through optimized code, although the current line will jump forward and back due to code scheduling. The values of variables printed from within `dbx` continue to be unreliable in optimized code.

Sun WorkShop 6

TABLE 3-6 lists the new features available with the release of Sun WorkShop 6. Some of the features are described in greater detail in the sections following the table.

TABLE 3-6 Sun WorkShop 6 New Features

Feature	Description
Text Editing	
Balloon expression evaluator	The balloon expression evaluator instantly shows you the current value of the expression at which your cursor is pointing in your text editor.
NEdit and Vim text editors	The NEdit and Vim text editors are new integrated text editors for this release.
Debugging a Program	
Button editor	The Button Editor now allows you to customize the toolbars in your editor window and the Debugging window, in addition to adding, removing, and editing buttons in the Custom Button window.

TABLE 3-6 Sun WorkShop 6 New Features *(Continued)*

Feature	Description
Debugging window	The Data History pane has been replaced with tabs for Data History and Program I/O, and an optional Data Display tab. The radio buttons for switching the Sessions and Threads panes have been replaced by tabs for Sessions, Threads, and Breakpoints.
Debugging Options dialog box	The Category list box has been replaced with tabs at the right side of the dialog box for switching among the categories of options.
New debugging options	You now choose to direct program input or output to the Program I/O tab in the Debugging window. You can now choose to have the call stack pop to the first debuggable source code if execution stops in a function in your program that is not debuggable source code. You can now choose to set the step granularity to Line, so that a single `next` command steps through a line regardless of the number of statements it includes. By default, the State, Stopped In, Evaluation Context, and number of sessions information is no longer displayed below the toolbar, but you can choose to display this information. You can choose to not have the debugger warn you if the `main ()` module of your program is not compiled with debugging information. You can choose to have the Data Display window shown as a tab in the Debugging window or as a separate window. You can delay the loading of modules compiled with `-xs` until the debugging information for these modules is needed, rather than having the modules loaded during the startup of the debugging session. If you want to debug processes with a large number (up to 300) LWPs (lightweight processes), you can now set debugging to be conservative in its resource usage when doing so. You can choose not to have the debugger check that `dbx` has exclusive control of the program being debugged. You can choose to allow the debugger to exclude certain shared libraries that are critical to `dbx` functionality.
Removed debugging option	The option to allow watchpoints in multithreaded programs has been removed from the Debugging Options dialog box.

TABLE 3-6 Sun WorkShop 6 New Features *(Continued)*

Feature	Description
Working With Projects	
Projects and worksets	This release of Sun WorkShop uses projects to track the files, programs, and targets associated with your development projects and to build your programs without your needing to write a makefile.
New Man Pages	
`makeprd`(1)	Sun WorkShop project file builder
`nedit`(1)	Motif user interface style text editor
`vim`(1)	Vi improved; a text editor for programmers
`xemacs`(1)	Emacs: The Next Generation

Text Editing

The following two sections describe the new Sun WorkShop 6 text editing features.

Balloon Expression Evaluator

The balloon expression evaluator instantly shows you the current value of the expression at which your cursor is pointing in your text editor. You can also see the type of expression and dereference the pointer through the balloon expression evaluator feature. For more information, see "Using the Balloon Expression Evaluator" in the Text Editing section of the online help.

NEdit and Vim Text Editors

Text editors are the center of the Sun WorkShop integrated development tool set that includes building, debugging, and browsing. The Sun WorkShop programming environment makes it possible to evaluate expressions, set breakpoints, and step through functions from your text editor.

NEdit, a plain-text editor with a graphical user interface for X/Motif systems, and Vim, an improved version of the vi standard text editor on UNIX systems, are new integrated text editors in this release. The following is a list of integrated text editors provided with Sun WorkShop 6:

- NEdit
- XEmacs
- GNU Emacs
- Vi
- Vim (with graphical user interface option)

For more information about each editor's options, see:

- The online documentation available from the Help menu in the editor's menu bar
- "Text Editor Options Dialog Box" in the Text Editing section of the online help

Debugging a Program

The following sections describe new program debugging features.

Button Editor

The Button Editor now allows you to customize the toolbars in your editor window and the Debugging window, in addition to adding, removing, and editing buttons in the Custom Button window.

For more information, see "Toolbar Options" in the Using the Debugging Window section of the online help.

Debugging Window

The following sections describe changes made in the Debugging window.

Session Status and Context Information

By default, the State, Stopped In, Evaluation Context, and Number of Sessions information no longer appears below the toolbar. If you want to display this information, you can select Show 3 Line Context/Status Area at Top of Display on the Window Layout tab in the Debugging Options dialog box. For more information, see "Displaying Session Status and Context Information" in the Using the Debugging Window section of the online help.

Data History Pane

The Data History pane has been replaced with tabs for Data History and Program I/O, and an optional Data Display tab:

- The Data History tab displays the Data History pane. For more information, see "Data History Tab" in the Using the Debugging WIndow section of the online help.
- The Program I/O tab displays program input and output within the Debugging window rather than in a separate Program Input/Output window. (For more information, see "Program I/O Tab" in the Using the Debugging Window section of the online help.) This is now the default behavior, but you can use the Program output section of the Debugging Output tab of the Debugging Options dialog box to choose where you want to direct program input and output. For more information, see "Redirecting a Program's Input/Output" in the Using the Debugging Window section of the online help.
- The Data Display tab is included (instead of a separate Data Display window) if you select Tab in Debug Window in the Data Display Window Shown As section on the Data Display Window tab in the Debugging Options dialog box. For more information, see "Data Display Tab" and "Choosing How to Show the Data Display" in the Using the Debugging Window section of the online help.

Sessions and Threads Panes

The radio buttons for switching the Sessions and Threads panes have been replaced by tabs for Sessions, Threads, and Breakpoints:

- The Sessions tab displays the Sessions pane. When you press the right mouse button over this tab, a popup menu is displayed. For more information, see "Sessions Tab" in the Using the Debugging Window section of the online help.
- The Threads tab displays the Threads pane. For more information, see "Threads Tab" in the Using the Debugging Window section of the online help.
- The Breakpoints tab displays a scrolling list of breakpoints and tracepoints assigned in your program. When you press the right mouse button over this tab, a popup menu is displayed that lets you enable, disable, delete and show source instantly for each breakpoint. The popup menu includes an Add item that displays the separate Breakpoints window in which you add, enable, disable, change, and delete breakpoints. For more information, see "Breakpoints Tab" in the Using the Debugging Window section of the online help.

Debugging Options Dialog Box

The Debugging Options dialog box contains the following new features.

Category Tabs

The Category list box in the Debugging Options dialog box has been replaced with tabs at the right side of the window for switching among the categories of options.

New Program Output Option

You now direct program input or output to the Program I/O tab in the Debugging window. As before, you can also direct it to a separate Program I/O window, to the `dbx` Commands window, or to a custom `pty`. For more information, see "Redirecting a Program's Input/Output" in the Using the Debugging Window section of the online help.

New Call Stack Option

During debugging, if your program was not compiled with `-g`, execution might stop in a function in the program that is not debuggable source code. You can now choose to have the call stack pop to the first debuggable source code in this case. For more information, see "Going Up the Stack When Execution Stops" in the Using the Debugging Window section of the online help.

New Stepping Option

By default, the step granularity for debugging is set to Statement, so that if more than one statement is included in a source code line, it requires the same number of commands to step through that line. You can now choose to set the step granularity to Line, so that a single next command steps through a line regardless of the number of statements it includes. For more information, see "Setting Step Granularity" in the Using the Debugging Window section of the online help.

New Window Layout Option

By default, the State, Stopped In, Evaluation Context, and number of sessions information is no longer displayed below the toolbar. If you want to display this information, you can select Show 3 Line Context/Status Area at Top of Display on the Window Layout tab in the Debugging Options dialog box. For more information, see "Displaying Session Status and Context Information" in the Using the Debugging Window section of the online help.

New Window Behavior Option

By default, the debugger warns you if the `main ()` module of your program is not compiled with debugging information. You can choose to not have this warning displayed. For more information, see "Being Warned If Your `main ()` Module is Not Compiled With Debugging Information" in the Using the Debugging Window section of the online help.

New Data Display Option

You can choose to have the Data Display window shown as a tab in the Debugging window or as a separate window. For more information, see "Choosing How to Show the Data Display" in the Using the Debugging Window section of the online help.

New Debugging Performance Option

You can delay the loading of modules compiled with `-xs` until the debugging information for these modules is needed, rather than having the modules loaded during the startup of the debugging session. This may shorten the debugging startup time when modules have been compiled with `-xs`. By default, this debugging option is set to on. For more information, see "Delaying Loading of Modules Compiled with `-xs`" in the Using the Debugging Window section of the online help.

New Forks and Threads Option

If you want to debug processes with a large number (up to 300) of LWPs (lightweight processes), you can now set debugging to be conservative in its resource usage. When doing so, debugging performance might be slowed. For more information, see "Debugging a Large Number of LWPs" in the Using the Debugging Window section of the online help.

New Advanced Options

By default, the debugger checks that `dbx` has exclusive control of the program being debugged. It prevents `dbx` from attaching to the process if another tool is already attached to the process. You can now turn off this behavior. For more information, see "Checking that `dbx` has Exclusive Control of the Program" in the Using the Debugging Window section of the online help.

By default, the debugger disallows the exclusion of certain shared libraries that are critical to `dbx` functionality. You can now choose to allow these libraries to be excluded, in which case you can debug core files only. For more information, see "Requiring Inclusion of Critical `dbx` Libraries" in the Using the Debugging Window section of the online help.

Working With Projects

This release of Sun WorkShop uses projects to track the files, programs, and targets associated with your development projects and to build your programs without your needing to write a makefile. A project is a list that includes the files and the compiler, debugger, and build-related options used to build an executable, a static library or archive, a shared library, a Fortran application, a complex application, or a user makefile application.

In previous versions of Sun WorkShop, a workset was used instead of a project. Projects still have some workset features (there continue to be menu picklists that make it easy to access directories and files associated with a project), but a project includes more information about your program, such as which source files you want to build and how you want them built. If you have Sun WorkShop worksets, you can automatically convert your worksets to Sun WorkShop 6 projects when you load them. For more information, see "Converting a Workset to a Project" in the Working With Projects section of the online help.

When you start Sun WorkShop, the Welcome to Sun WorkShop dialog box opens and gives you immediate access to Sun WorkShop projects and the project wizard. Sun WorkShop also has project functions available from the Project Menu in the WorkShop Main Window to help you complete the following tasks:

- Create a new project or build a simple program using the project wizard and your own makefile or a makefile that Sun WorkShop creates for you (see "Creating a New Project" in the Working With Projects section of the online help)
- Change existing project settings, including how you want your project compiled and whether you want source browsing information generated (see "Editing a Project" and "Edit Current Project Window" in the Working With Projects section of the online help)

You can also choose to use the Sun WorkShop 6 programming environment without loading a project. Picklists keep track of the files, programs, directories, and targets associated with your development projects (see "WorkShop Targets" in the Building Programs section of the online help for more information). You can access each file, build the target, and debug the executable from the menus in the WorkShop Main Window. Build target information cannot be edited because it is not persistent; rather, it changes as you access, add, and remove build targets.

Sun WorkShop TeamWare 6

TABLE 3-7 lists the new features available with the release of Sun WorkShop™ TeamWare 6. Some of the features are described in greater detail in the sections following the table.

TABLE 3-7 Sun WorkShop TeamWare 6 New Features

Feature	Description
Autofreezepointing	Autofreezepointing creates a freezepoint file for you before or after specific transactions. You can select the time that you want freezepoint files created: before or after bringovers, putbacks, undo actions, or resolve actions.
Versioning customized menu	The customized menu feature adds a new menu in the Versioning window titled "Customized," which provides access to your own commands. For information about creating a customized menu, see "Creating a Customized Menu" in the Managing Files section of the online help.
Delta comments	This new option adds delta comments to transaction output and email notification, including delta number, owner, and comments. In Configuring, choose Workspace ➤ Bringover Create/Bringover Update/Putback and check the Delta Comments box, or use the `-d` option with the bringover or putback command.
Merging diff navigator	The diff navigator appears between two unmerged files. You can click on the slide boxes on either side of the diff navigator to scroll through either file, or click on the arrows on the top or bottom to move the same distance in both files.
Configuring Menu Reorganization	Sun Workshop 6 TeamWare has implemented changes to the Configuring user interface.

TABLE 3-7 Sun WorkShop TeamWare 6 New Features *(Continued)*

Feature	Description
Putback Validation	When you turn on putback validation, only putbacks are allowed to the workspace. You can control which users can perform a putback and require that they have a specific password.
SCCS Admin Flags	Allows you to set SCCS admin flags for a file.
Workspace integrity check	New option to `workspace` command: `check [ -W ] [ - s ] wsname ...` checks files, access modes, parent-child relationships, and condition of the history file. The command exits with the following values: 0 = workspace is okay or 1 = error.
Workspace History Viewer	Sun WorkShop TeamWare now includes an easy way to view the information in the workspace history file.
Workspace labels	With this feature, you can give a workspace a descriptive name that is more meaningful to your team. Choose Workspace ➤ Properties and select the Description tab.
New man page	`description`(4)

Configuring Menu Reorganization

Sun Workshop TeamWare 6 includes the following changes to the menus.

TABLE 3-8 Sun WorkShop TeamWare 6 Menu Changes

Sun WorkShop TeamWare 2.2	Sun WorkShop TeamWare 6
Configuring	
File ➤ Load Parent	Workspace ➤ Load Parent
File ➤ Load Children	Workspace ➤ Load Children
File ➤ Create Empty Child Workspace	Workspace ➤ Create Child
Edit ➤ Delete	Workspace ➤ Delete
Edit ➤ Rename	Workspace ➤ Rename
Edit ➤ Parent	Workspace ➤ Reparent
Edit ➤ Update ➤ Nametable	Workspace ➤ Update Nametable
Transactions ➤ Bringover ➤ Create	Actions ➤ Bringover Create

TABLE 3-8 Sun WorkShop TeamWare 6 Menu Changes *(Continued)*

Sun WorkShop TeamWare 2.2	Sun WorkShop TeamWare 6
Transactions ➤ Bringover ➤ Update	Actions ➤ Bringover Update
Options ➤ Workspace	Workspace ➤ Properties
Options ➤ Workspace ➤ Edit Locks	Workspace ➤ Edit Locks
NEW	View ➤ Refresh
NEW	Workspace ➤ Properties ➤ Freezepointing
NEW	Workspace ➤ Properties ➤ Putback Validation
NEW	Options ➤ Configuring ➤ Load Children: Selective/All
Versioning	
NEW	File ➤ File Info
NEW	Commands ➤ Uncheckout

Putback Validation

When you turn on putback validation, only putbacks are allowed to the workspace. The user is prompted for a password (Integration Request Identifier) before performing a putback. This feature only records the Integration Request Identifier, it does not check it. To check the Integration Request Identifier, you must write your own validation program. For more information, see "Protecting Workspaces With Putback Validation" in the Managing Workspaces section of the online help.

SCCS Admin Flags

You can set SCCS admin flags for a file by using Versioning, choose File ➤ File Info. For example, if you want Versioning to prompt for MRs (modification request strings) during putbacks, put the name of a validation program in the Validation Program box.

Workspace History Viewer

Sun WorkShop TeamWare now includes an easy way to view the information in the workspace history file (choose Workspace ➤ View History). In the Workspace History Viewer, you can view the transaction history of a workspace, transaction details, comments, and the command log. You can sort and filter the entries and search the comments and command log.

Sun WorkShop Visual 6

TABLE 3-9 lists the new features available with the release of Sun WorkShop Visual 6. Some of the features are described in greater detail in the sections following the table.

TABLE 3-9 Sun WorkShop Visual 6 New Features

Feature	Description
Swing Support	Sun WorkShop 6 Visual now has the ability to generate Java Swing code in addition to Java 1.0 and 1.1 code. As well as generating appropriate Swing components for the Motif widgets, Visual has increased the range of supported mappable resources: It is now possible to generate appropriate Swing code for toggles that contain images, shell icons, shell resize and delete response, the contents of lists, and rowcolumn entry alignment, among other extensions.
Enhanced Windows Support	Sun WorkShop Visual has added support for mapping X events to Windows MFC.
Integration With Sun WorkShop Projects	Sun WorkShop Visual works with Sun WorkShop project wizards to help create projects with graphical user interfaces.

Swing Support

Sun WorkShop 6 Visual now has the ability to generate Java™ Swing code in addition to Java 1.0 and 1.1 code. As well as generating appropriate Swing components for the Motif widgets, Sun WorkShop 6 Visual has increased the range of supported mappable resources: It is now possible to generate appropriate Swing code for toggles that contain images, shell icons, shell resize and delete response, the contents of lists, and rowcolumn entry alignment, among other extensions.

The MWT class library, which maps Motif components into Java where the standard classes lack equivalence, has been ported to Swing in order to provide a more consistent look and feel. The dependence on the MWT to provide a Motif-compatible interface has been reduced as some of the Motif components are now mapped directly into an appropriate Swing component.

The Java Layout emulation widgets have been reworked where necessary to provide a more consistent behavior with respect to Java Layout characteristics.

As part of a move towards supporting cross-platform code for third- party (non-Motif) components, Visual now has the ability to specify default base classing for any integrated component. The classing can be on a general language basis or for specific variants. For example, it is possible to specify in a general way the proposed default class for Java, as well as Java 1.0, Java 1.1, and Swing-specific classing. Proposed component MFC classing can also be specified. The mechanisms address the problem wherein each given third-party component required individual manual configuration in order to create the right kind of object in the target language. Third-party specific resources are not mapped. The mechanisms do not address the following situations:

- A given component needs to be mapped into multiple native objects
- A compound component can be mapped by consideration of the constituent built-in parts.

Enhanced Windows Support

Sun WorkShop Visual 6 has added the following support for mapping X events to Windows MFC.

TABLE 3-10 Sun WorkShop Visual Mappings for X Events to Windows MFC

MouseMotion	Generates a generic handler for any mouse movement with or without button press
ButtonPress	Generate all three handlers: Left, Center, and Right pressed handlers
ButtonRelease	Generate all three handlers: Left, Center, and Right release handlers
EnterWindow	`MouseActivate`
ExposureMask	`EraseBkgnd`
KeyPressMask	`WM_KEYDOWN`
KeyRelease	`WM_KEYUP`
KeymapstateMask	`WM_SYSKEYUP/WM_SYSKEYDOWN`

TABLE 3-10 Sun WorkShop Visual Mappings for X Events to Windows MFC *(Continued)*

LeaveWindowMask	`WM_KILL_FOCUS`
ResizeRedirect	`WM_SIZE`
PropertyChangeMask	`ON_WM_PAINT`
VisibilityChangeMask	`WM_SHOWWINDOW`

Sun Performance Library

Sun Performance Library™ is a set of optimized, high-speed mathematical subroutines for solving linear algebra and other numerically intensive problems. Sun Performance Library is based on a collection of public domain applications available from Netlib at `http://www.netlib.org`. These routines have been enhanced and bundled as the Sun Performance Library.

TABLE 3-11 lists the new features available with the Sun WorkShop 6 release of the Sun Performance Library. Some of the features are described in greater detail in the sections following the table.

TABLE 3-11 Sun Performance Library New Features

Feature	Description
Support for LAPACK 3.0	LAPACK 3.0 subroutines have been added. The previous version of Sun Performance Library was based on LAPACK 2.0. The current version of Sun Performance Library still maintains compatibility with LAPACK 2.0 and LAPACK 1.x.
Sparse solver package	The sparse solver package provides routines for solving sparse matrices (symmetric, structurally symmetric, and unsymmetric coefficient matrices) using direct methods and a choice of fill-reducing ordering algorithms, including user specified orderings.

TABLE 3-11 Sun Performance Library New Features *(Continued)*

Feature	Description
UltraSPARC III support	This release supports the UltraSPARC III instruction set architecture. To use the code that is specific to UltraSPARC III, compile with `-xarch=v8plusb` for 32-bit code or `-xarch=v9b` for 64-bit code.
Fortran 95 Language Feature Support	This release supports Fortran 95 language features. You can use the Sun Performance Library modules and definitions by including the `f95` statement `USE SUNPERF` in the code.
Changes to Sun Performance Library Licensing	Sun Performance Library is no longer licensed. However, you should continue to use `-xlic_lib=sunperf` to ensure that the application links with the correct support libraries, and to ensure the correct version of Sun Performance Library is selected.

Fortran 95 Language Feature Support

By including the `f95` statement `USE SUNPERF` in your application, you can use the Sun Performance Library modules and definitions with the following features:

- **Type independence.** In Fortran 77 routines, you must specify the type as part of the name. In Fortran 95, a routine for a specific data type can be determined by the data type of the arguments passed to the routine.
- **Compile-time checking.** In Fortran 77, it is generally impossible for the compiler to know what parameters should be passed to a particular routine. In Fortran 95, the `USE SUNPERF` statement allows the compiler to know what the number, type, size, and shape of each parameter to each Sun Performance Library routine should be. It can check your calls against the expected value and identify errors during compilation.
- **Optional parameters.** In Fortran 77, all parameters must be specified in order for all routines. Fortran 95 allows some parameters to be optional. In Sun Performance Library, all increment parameters (`INCX`, `INCY`, and so on), workspaces, leading dimensions (`LDA`, `LDB`, and so on), and length or size parameters are optional.

For information on using these features and examples, see the *Sun Performance Library User's Guide*.

Changes to Sun Performance Library Licensing

Sun Performance Library is no longer licensed. However, you should continue to link using `-xlic_lib=sunperf` rather than `-lsunperf`. Use `-xlic_lib=sunperf` to ensure that the following occurs:

- The application is linked with the correct support libraries. In this release, Sun Performance Library is compiled with Fortran 95 instead of Fortran 77. Using `-xlic_lib` will correctly link in the Fortran 95 runtime libraries instead of the Fortran 77 runtime libraries.
- The correct version of Sun Performance Library is used. There are different versions of Sun Performance Library to support programs built with and without `-subparallel` and for different values of `-xarch`. Using `-xlic_lib=sunperf` will cause the driver to use the version of Sun Performance Library that best matches your command line options.

Sampling Analyzer

Sun WorkShop 6 Sampling Analyzer is a complete rewrite of the Analyzer that was provided with Sun WorkShop 5.0.

TABLE 3-12 lists the new features available with the release of the Sun WorkShop 6 Sampling Analyzer. Some of these features are described in greater detail in the sections following the table. See also "Additional Changes" on page 74.

TABLE 3-12 Sampling Analyzer New Features

Feature	Description
Function List as Primary Display	The Function List is the primary display and is displayed by default when the Analyzer is invoked.
Multiple Metrics	The Function List displays multiple metrics at the same time, instead of requiring you to select one category at a time to view. The Function List can also display metrics as values or a percentage.
Summary Metrics Window	A new Summary Metrics window, accessed from the View menu, displays all metrics recorded for a selected function, both as values and percentages. The contents of the Summary Metrics window are independent of what appears in the function list display.

TABLE 3-12 Sampling Analyzer New Features *(Continued)*

Feature	Description
Callers-Callees Window	From the Function List, you can access a new Callers-Callees window that shows how metrics are attributed from the callees of a selected function and to the callers of that function.
Generate Annotated Source Code	You can now generate annotated source code for a selected function and display the results in an edit window.
Generate Annotated Disassembly	You can generate annotated disassembly for the selected function and display the results in an edit window.
Filter Data by Samples, Threads, LWPs	You can now use the Select Filters dialog box to filter data by samples, threads, LWPs, or any combination of these. All displays and windows are updated to show data from the selected subset only.
Thread Synchronization Delay Metrics	Two thread synchronization delay metrics are now available: a count of synchronization events exceeding the designated threshold, and the aggregate delay from those events. For more information about metrics, see "Metrics" on page 74.
Load Multiple Experiments	You can now load multiple experiments into the Analyzer at the same time. Their combined metrics appear in the Function List display.

Function List as Primary Display

The Function List is the primary display, and is displayed by default when the Analyzer is invoked.

Because the Function List can display multiple types of metrics at the same time, the Data list option menu has been redesigned to change only the display type. The Display list option menu has been removed.

You can now show metrics in the Function List as absolute values in seconds or counts, or a percentage of the total program metric, or both. By invoking a Select Metrics dialog box from the Function List display, you can do any of the following:

- Select the metrics displayed in the Function List
- Display metrics as counts, percentages, or both
- Specify which metric is used to sort the Function List
- Reorder the list

Callers-Callees Window

From the Function List, you can access a new Callers-Callees window that shows how metrics are attributed from the callees of a function and to the callers of that function. The Caller-Callees window shows the selected function in the center of the display, with callers of that function in the panel above and callees of the function in the panel below. For the selected function, the attributed metric represents usage within the function itself. For the callers above, it represents usage within the selected function and all functions it calls, as attributed up the callstack to its callers. For the callees below, it represents the proportion of the callee's metric that is attributable to calls from the selected function.

You can navigate through the program's structure in the Callers-Callees window by clicking on a function in either the caller panel or the callee panel; the display recenters on the newly selected function.

Generate Annotated Source Code

You can now generate annotated source code for a selected function and display the results in an edit window. Source code is annotated with per-line metrics, using the same set of metrics as the Function List. The source code also contains compiler parallelization commentary and Fortran 95 copyin and copyout commentary interleaved with the source. Source code display requires compiling with `-g` and will work for optimized code (`-g` no longer disables optimizations and parallelization).

Generate Annotated Disassembly

You can now generate annotated disassembly for a selected function and display the results in an edit window. Disassembly is annotated with per-instruction metrics, using the same set of metrics as the Function Display. Disassembly also contains compiler commentary and interleaved source code.

Metrics

The following new metrics or changes to existing metrics are provided with Sun WorkShop 6 Analyzer:

- Execution profile data is now called *clock-based profiling*. Execution profile data always includes called-function times.
- Histogram data is now called *exclusive metrics*.
- Cumulative data metrics are now called *inclusive metrics*.
- Clock-based profile data generates the following metrics:
  - Total LWP time
  - User CPU time
  - System CPU time
  - System wait time
  - Text-page fault time
  - Data-page fault time
  - Wall-clock time
- Thread synchronization delay tracing, a new kind of data, is provided that generates the following metrics:
  - A count of synchronization events exceeding the designated threshold
  - The aggregate delay from those events.

  These metrics are not collected by default.

Additional Changes

The Sun WorkShop 6 Sampling Analyzer provides the following additional changes:

- You can no longer select a subset of samples by clicking on them in the Overview display. The Select Samples text box and arrow buttons are no longer part of the Analyzer, and selection commands no longer appear in the View menu. You must do all filter selection of samples, threads, and LWPs in the Select Filters dialog box.
- You cannot delete an experiment record from inside the Analyzer. The Experiment ➤ Delete command has been changed to Experiment ➤ Drop. Use this command to drop an experiment from the Analyzer. The experiment record remains on disk until you remove it with the `er_rm` command.
- The format of `er_export` ASCII data has been changed. (`er_export` is used only for debugging.)

Installation

TABLE 3-13 lists the new installation features available with the release of Sun WorkShop 6.

TABLE 3-13 Installation New Features

Feature	Description
Web Start	Web Start is the new installation software.
FLEXlm 7.0b	FLEXlm 7.0b is the license manager software provided with the Sun WorkShop 6 programming environment.

Documentation in HTML

The manuals, man pages, and readme files in Sun WorkShop 6 and Sun WorkShop TeamWare 6 are available in HTML as well as in text files. The online help is now in HTML.

To view Sun WorkShop 6 documentation that is available in HTML format, you must use Netscape™ Communicator 4.0 or a compatible Netscape version. Netscape Communicator is included in the Solaris™ 7 Operating Environment and the Solaris™ 8 Operating Environment.

If you are running the Solaris 2.6 Operating Environment and you do not have Netscape Communicator 4.0 or a compatible version, you can download Netscape Communicator 4.7 from the following Netscape Communications Corporation Web site:

```
http://www.netscape.com/download/index.html
```

Sun WorkShop online help (in HTML) requires that you have JavaScript™ enabled, which is a setting in Netscape preferences.

To access the installed Sun WorkShop 6 documentation in HTML format, point your browser at the following file:

```
file:/opt/SUNWspro/docs/index.html
```

If your Sun WorkShop software is not installed in the `/opt` directory, contact your system administrator for the equivalent path on your system.