What's New In The Oracle® Solaris Studio 12.2 Release

Part No: 821–2414
January 2011

110606@25097

Contents

Preface

This guide describes the new and changed features in the *Oracle Solaris Studio 12.2* release.

Oracle Solaris Studio is now the new name for the Sun Studio compilers and tools software collection. Releases prior to this release retain the *Sun Studio* name.

Supported Platforms

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

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

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

For supported systems, see the hardware compatibility lists.

Accessing Solaris Studio Documentation

You can access the documentation at the following locations:

- The documentation is available from the documentation index page at `http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation`.
- Online help for all components of the IDE, the Performance Analyzer, dbxtool, and DLight, is available through the Help menu, as well as through the F1 key and Help buttons on many windows and dialog boxes, in these tools.

Documentation in Accessible Formats

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

Type of Documentation	Format and Location of Accessible Version
Manuals	HTML from the Oracle Solaris Studio 12.2 collection on docs.sun.com
What's New in The Oracle Solaris Studio 12.2 Release (formerly the component README files)	HTML from the Oracle Solaris Studio 12.2 collection on docs.sun.com
Man pages	Displayed in an Oracle Solaris terminal using the `man` command
Online help	HTML available through the Help menu, Help buttons, and F1 key in the IDE, dbxtool, DLight, and the Performance Analyzer
Release notes	HTML from the Oracle Solaris Studio 12.2 collection on docs.sun.com

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Oracle is not responsible for the availability of third-party web sites mentioned in this document. Oracle does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Oracle will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Resources for Developers

Visit `http://www.oracle.com/technetwork/server-storage/solarisstudio` to find these frequently updated resources:

- Articles on programming techniques and best practices
- Documentation of the software, as well as corrections to the documentation that is installed with your software
- Tutorials that take you step-by-step through development tasks using Oracle Solaris Studio tools

- Information on support levels
- User forums at `http://forums.sun.com/category.jspa?categoryID=113`

Typographic Conventions

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

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for shells that are included in the Oracle Solaris OS. Note that the default system prompt that is displayed in command examples varies, depending on the Oracle Solaris release.

TABLE P–2 Shell Prompts

Shell	Prompt
Bash shell, Korn shell, and Bourne shell	`$`
Bash shell, Korn shell, and Bourne shell for superuser	`#`
C shell	`machine_name%`

TABLE P–2 Shell Prompts *(Continued)*

Shell	Prompt
C shell for superuser	`machine_name#`

Documentation, Support, and Training

See the following web sites for additional resources:

- Documentation (`http://docs.sun.com`)
- Support (`http://www.oracle.com/us/support/systems/index.html`)
- Training (`http://education.oracle.com`) – Click the Sun link in the left navigation bar.

Oracle Welcomes Your Comments

Oracle welcomes your comments and suggestions on the quality and usefulness of its documentation. If you find any errors or have any other suggestions for improvement, go to `http://docs.sun.com` and click Feedback. Indicate the title and part number of the documentation along with the chapter, section, and page number, if available. Please let us know if you want a reply.

Oracle Technology Network (`http://www.oracle.com/technetwork/index.html`) offers a range of resources related to Oracle software:

- Discuss technical problems and solutions on the Discussion Forums (`http://forums.oracle.com`).
- Get hands-on step-by-step tutorials with Oracle By Example (`http://www.oracle.com/technology/obe/start/index.html`).
- Download Sample Code (`http://www.oracle.com/technology/sample_code/index.html`).

CHAPTER 1

Introducing The Oracle Solaris Studio 12.2 Release

The release of Oracle Solaris Studio offers a number of new and changed features as outlined in this *What's New...* Guide. This Guide replaces the component README files published in previous releases on the Sun Developer Network portal.

The most significant change, of course, is the name: Sun Studio is *Oracle Solaris Studio.*

What Is Oracle Solaris Studio?

Oracle Solaris Studio comprises a suite of tools for application development on Solaris and Linux operating environments:

- High performance optimizing compilers and runtime libraries for C, C++, and Fortran (`cc`, `CC`, and `f95`) that natively implement the OpenMP 3.0 API for shared memory parallelization
- The scriptable and multithread aware interactive dbx debugger and dbxtool debugger GUI
- New tools for discovering memory leaks and code coverage, `discover` and `uncover`
- The highly optimized and multithreaded Sun Performance Library
- A Performance Analyzer to profile single- and multi-threaded applications to detect performance bottlenecks and inefficiencies, and DLight for system profiling using DTrace technology on Solaris environments.
- A Thread Analyzer to identify potential and hard-to-detect data race and deadlock conditions at runtime before they occur in multithreaded applications.
- An IDE tailored for use with the component compilers, debugger, and analysis tools, along with a code-aware editor, workflow, and project functionality for building applications.

Links to the complete set of Oracle Solaris Studio documentation can be found on the Oracle Technical Network portal, `http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation`

About This *What's New...* Guide

This Guide is organized into separate chapters for the Compilers, Libraries, Peformance Analysis Tools, Debugging Tools, the IDE, and other related tools. A chapter on known problems, limitations, and workarounds outlines additional information regarding Oracle Solaris Studio 12.2 tools.

To keep up to date with the latest information regarding this release, go to the Solaris Studio portal on the Oracle Technical Network.

CHAPTER 2

Compilers

This chapter describes the new and changed compiler features in this Oracle Solaris Studio release.

New/Changed Features Common To The Compilers

The following lists the significant changes common to the C, C++, and Fortran compilers since the previous release. Details can be found in the compiler man pages and user guides.

- The compilers support the SPARC VIS3 version of the SPARC-V9 ISA. Compiling with the `-xarch=sparcvis3` option enables the compiler to use instructions from the SPARC-V9 instruction set, plus the UltraSPARC extensions, including the Visual Instruction Set (VIS) version 1.0, the UltraSPARC-III extensions, including the Visual Instruction Set (VIS) version 2.0 the fused multiply-add instructions, and the Visual Instruction Set (VIS) version 3.0.
- The default for the `-xvector` option has changed on x86-based systems to `-xvector=simd`. Streaming extensions are used on x86-based systems by default where beneficial at optimization level 3 and above. The suboption `no%simd` can be used to disable it. On SPARC-based systems, the default is `-xvector=%none`
- Support for the AMD SSE4a instruction set is now available. Compile with the `-xarch=amdsse4a` option.
- The man pages have been updated with the correct expansion for `-xtarget` values `ultra3`, `ultra3i`, `ultra3cu`, `ultra4`, and `ultra4plus`.
- The new `-traceback` option enables an executable to print a stack trace if a severe error occurs. This option causes the executable to trap a set of signals and print a stack trace and core dump before exiting. If multiple threads generate a signal, a stack trace will be produced only for the first one. To use traceback, add the `-traceback` option when linking a program with either `f95`, `cc`, or `CC`. For convenience, the option is also accepted at compile-time but is ignored. Using the `-traceback` option with the `-G` option to create a shared library is an error. See the compiler man pages for details on the `-traceback` option

- The `-mt` option has been changed to `-mt=yes` or `-mt=no`. The `-mt=yes` option assures that libraries are linked in the appropriate order. See the compiler man pages for details.
- New pragmas have been added for C and C++. For details, see the compiler user guides.
- The `#warning` compiler directive (C and C++) issues the text in the directive as a warning and continues compilation
- The (C and C++) header file `mbarrier.h` is now available. It defines various memory barrier intrinsics for multithreaded code on SPARC and x86 processors. For details, see the compiler user guides for details.
- The `-xprofile=tcov[:`*prof_dir*`]` option accepts an optional profile directory pathname argument. If a profile directory pathname is specified, the compiled program generates data that can be used by either `tcov(1)` or feedback compilation with `-xprofile=use:`*prof_dir*. For details, see the compiler user guides.
- In this release, the dependency file written by the `-xMD` and `-xMMD` options (C/C++) overwrites any previously existing file. The file name is derived from the `-o` *filename*, if specified, or the input source filename, with a `.d` suffix appended, or the filename specified by the `-xMF` option. When the `-o` *filename* or `-xMF` *filename* is specified with the `-xMD` or `-xMMD` option, only a single source file is accepted. Compiling multiple source files this way is an error.

C Compiler

The following lists the new and changed features in this release of version 5.11 of the C compiler. For details, see the *Oracle Solaris Studio 12.2: C User's Guide* and the `cc` man page.

- A change to the C compiler corrects the way a `struct` containing complex types is passed and returned on SPARC processors in 64-bit mode. Previously, these `struct` values were sometimes passed and returned in the wrong registers, creating binaries that were incompatible with binaries created by the `gcc` compiler. Because this change affects elements of the existing ABI as implemented in the Solaris Studio C compiler, if any source file in an application uses structs with complex fileds, *the entire source base for the application must be recompiled to avoid the possibility of wrong answers*. Compiling for 32-bit SPARC processors, and 32-bit or 64-bit x86 processors, is not affected by this change.

C++ Compiler

The following lists the new and changed features in this release of version 5.11 of the C++ compiler. For details, see the *Oracle Solaris Studio 12.2: C++ User's Guide* and the `CC` man page.

- The `-g` option with any `-O` or `-xO` option but without the `+` option, produces inlining. Examples:
  - `CC -g foo.cc` produces a debuggable `a.out` with no inlining

    - `CC -g -O foo.cc` produces a debuggable `a.out` with inlining
    - `CC -g0 foo.cc` produces a debuggable `a.out` with inlining
- The C++ option `-xalias_level=compatible` asserts that the program meets the requirements of the C++ standard.
- Support has been added for the Apache C++ library installed in Oracle Solaris.
- The `-compat=g` option adds certain `gcc` compatabilities.
- The `-features=[no%]rvalueref` option restores the compiler's handling of non-`const` references to a temporary or rvalue.

Fortran Compiler

The following lists the new and changed features in this release of version 8.5 of the Fortran compiler. For details, see the *Oracle Solaris Studio 12.2: Fortran User's Guide* and the `f95` man page.

- The new **`-xkeepframe`**[**`=`**[**`%all,%none`**, *name*, **`no%`***name*] option prohibits stack related optimizations for the named functions. **`%all`** prohibits stack related optimizations for all the code. **`%none`** allows stack related optimizations for all the code. The default is **`-xkeepframe=%none`**.
- Additional features from the F2003 Fortran standard have been implemented.
- The **`IVDEP`** directive tells the compiler to ignore some or all loop-carried dependences on array references that it finds in a loop for purposes of optimization. This enables the compiler to perform various loop optimizations that would not otherwise be possible. The **`-xivdep`** option can be used to disable the **`IVDEP`** directives, or to determine how the directives should be interpreted.

OpenMP

The following lists the new and changed features for the OpenMP 3.0 shared memory API implemented by the C, C++, and Fortran compilers in this release. For details, see the *Oracle Solaris Studio 12.2: OpenMP API User's Guide*.

- Support for OpenMP debugging in the dbx Debugger. The following enhancements have been made to dbx:
    - New commands for displaying information on OpenMP regions, tasks, and thread sets.
    - Extensions to the `print –s`, `thread –info`, `whatis`, and `where` commands
    - New OpenMP synchronization events.
- Autoscoping has been extended to task regions. This feature relieves the programmer of explicitly determining the scopes of variables in a parallel or a task region. The compiler determines the scopes of variables by analyzing the code and applying some smart rules.

- The new `SUNW_MP_WAIT_POLICY` environment variable improves the waiting behavior of threads in the program, and allows the programmer to have fine control over the behavior of threads that are waiting for work (idle), waiting at a barrier, or waiting at `taskwait`.
- New functionality has been added to the `SUNW_MP_WARN` OpenMP environment variable: In addition to controlling warning messages issued by the OpenMP runtime library, when `SUNW_MP_WARN` is set to `TRUE`, the runtime library outputs the settings of all environment variables for informational purposes, including environment variables that are explicitly set by the user, as well as those that are set by default by the library.
- The behavior controlled by the `SUNW_MP_PROCBIND` environment variable has changed on Oracle Solaris platforms: Setting `SUNW_MP_PROCBIND` to `TRUE` binds the main thread to the processor it is running on at the binding moment. The binding moment is the first encounter of a parallel region or the first call to a OpenMP runtime routine such as `omp_set_num_threads()`. The slave threads are bound in a round-robin fashion starting from the processor that the main thread is bound to.
- Use the Thread Analyzer tool to detect data races and deadlocks in an OpenMP program. In this release, the Thread Analyzer functionality has been extended to detect data races in binaries without the need to recompile. Refer to the *Oracle Solaris Studio 12.2: Thread Analyzer User's Guide* for details.

CHAPTER 3

Libraries

This chapter describes the new and changed features relating to the libraries in this Oracle Solaris Studio release.

Math Libraries

The math library `libcx` is now obsolete. Libraries `libm9x.so.0`, `libmvec.a`, and `libmvec_mt.a` are also obsolete and have been removed in this release.

Sun Performance Library

About the Sun Performance Library

This release of Sun Performance Library is available on the Solaris Operating System version 10. It is also available on several Linux operating environments.

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 subroutines available from Netlib at `http://www.netlib.org/` that have been enhanced and optimized, and bundled together as the Sun Performance Library. It includes the following standard libraries:

- LAPACK version 3.1.1 for solving linear algebra problems.
- BLAS1 (Basic Linear Algebra Subprograms) for performing vector-vector operations.
- BLAS2 for performing matrix-vector operations.
- BLAS3 for performing matrix-matrix operations.
- Netlib Sparse-BLAS for performing sparse vector operations.

- NIST Fortran Sparse BLAS version 0.5 for performing fundamental sparse matrix operations.
- SuperLU version 3.0 for solving sparse linear systems of equations.

Sun Performance Library includes the following additional routines:

- Fast Fourier transform (FFT) routines
- Direct Sparse Solver routines

Compatibility

The LAPACK 3.1.1 routines in Sun Performance Library are compatible with the user routines from previous versions of LAPACK, including 1.x, 2.0, and 3.0, and with all routines in LAPACK 3.1.1. However, due to internal changes in LAPACK 3.1.1, compatibility with internal routines cannot be guaranteed.

Internal routines that might be incompatible are called auxiliary routines in the LAPACK source code available from Netlib. Some information on auxiliary routines is included in the LAPACK Users' Guide, available from the Society for Industrial and Applied Mathematics (SIAM) at `http://www.siam.org/`.

Because the user interfaces to the LAPACK auxiliary routines can change from release to release of LAPACK, the user interfaces to the LAPACK auxiliary routines in Sun Performance Library can change as well. Auxiliary routines compatible with LAPACK 3.1.1 are generally available for users to call; however, the auxiliary routines are not specifically documented, tested, or supported. Be aware that the user interfaces for the LAPACK auxiliary routines can change in future releases of Sun Performance Library, so that the user interfaces comply with the version of LAPACK supported by that version of the Sun Performance Library.

Documentation

The following Sun Performance Library documentation is available:

- Man pages (section 3p) for each function and subroutine in the library
- The *Oracle Solaris Studio 12.2: Sun Performance Library User's Guide* describes and shows examples for:
  - Using Sun Performance Library routines
  - Using the Fortran and C interfaces
  - Using optimization and parallelization options
  - Using SPSOLVE and SuperLU sparse solver packages
  - Using the FFT routines

For additional reference information, see the *LAPACK Users' Guide 3rd ed.*, by Anderson, E. and others, SIAM, 1999, which is available from the Society for Industrial and Applied Mathematics (SIAM) or your local bookstore. The *LAPACK Users' Guide* is the official reference for the base LAPACK 3.1.1 routines available on Netlib and provides mathematical descriptions of the LAPACK 3.1.1 routines.

New and Changed Features in This Release

- In this release, linking with Sun Performance Library has changed for the Fortran and C compilers. Now Fortran, C, and C++ all use the `-library=sunperf` option instead of the `-xlic_lib=sunperf` option. If you want to link statically, add the `-staticlib=sunperf` option after the `-library=sunperf` option.
- `libsuniperf` (IBLAS) is classified Obsolete and is removed from Oracle Solaris Studio in this release.

New and Changed Features Introduced in the Previous Release

The Sun Studio 12 update 1 release introduced the following in the Sun Performance Library:

- Sun Performance Library now includes the ScaLAPACK 1.8.0 high performance cluster library. This library works with Sun HPC ClusterTools 8.1 based on the OpenMPI 1.3 release. The reference implementation along with documentation can be found at `http://www.netlib.org/scalapack/`.
- The new Custom Library Tool provides the option to create scaled down versions of Sun Performance Library. The Custom Library Tool, `gen_custom`, extracts routines from an archive library and then recombines them into a customized library. This can reduce the size footprint of a large library like the Sun Performance Library to just those routines the user needs. See the `gen_custom`(3p) man page for further information.
- Numerous performance improvements have been made for BLAS, LAPACK, and FFT routines.
- Support for Intel(R) CoreTM i7 (Nehalem) and AMD Quad-Core OpteronTM (Shanghai) CPUs is available. To link with this library, use the following options:

  `-m64 -xlic_lib=sunperf` (C and Fortran)

  `-m64 -library=sunperf` (C++)
- Support for Fujitsu SPARC64-VII(R) CPUs is available. This version of Sun Performance Library uses the floating point multiply-add instruction to achieve the best performance possible. To link with this library, use the following options:

  `-xtarget=sparc64vii -fma=fused -xlic_lib=sunperf` (C and Fortran)

  `-xtarget=sparc64vii -fma=fused -library=sunperf` (C++)
- ZGEMM improvements for SPARC64-VI and SPARC64-VII
- LAPACK routines are updated to conform to the latest specification of LAPACK 3.1.1
- Support for Woodcrest CPUs is available.
- Support for SPARC64-VI CPUs is available.

For x86-based systems:

- Libraries are available on 32-bit and 64-bit systems with SuSE Linux Enterprise Server 9 or Redhat Enterprise Linux 4 operating environment.
- Routines with 64-bit integer parameters are now available. That is, `DAXPY()` and `DAXPY_64()` are in all versions of the Sun Performance Library.
- Serial version of the sparse solver package SuperLU is available and can be called from C drivers or through the existing Fortran-based sparse solver in the Library.
- At this time, quad-precision routines (`dqdoti, dqdota`) are not available
- Interval BLAS routines are available for Solaris OS and Linux OS on SSE2-enabled and above x86 systems.

For SPARC Processors:

- BLAS and FFT improvements for the UltraSPARC IV+ and UltraSPARC IV processors are included.
- Support for SPARC64VI CPUs is available. This version of Sun Performance Library uses the floating point multiply-add instruction to achieve the best performance possible on SPARC64VI CPUs. To link with this library, compile/link with the `-xtarget=sparcfmaf` flag
- Serial version of the sparse solver package SuperLU is available and can be called from C drivers or through the existing Fortran-based SPSOLVE sparse solver in the Library.

CHAPTER 4

Performance Analysis Tools

This chapter describes the new and changed features in the performance analysis tools in this Oracle Solaris Studio release.

The Performance Analyzer

This section describes the new and changed features in this release of the Solaris Studio Performance Analyzer and related tools. For details, see the *Oracle Solaris Studio 12.2: Performance Analyzer* manual.

Changes to Experiment Format

The experiment format has been extended, but the version number is currently unchanged (10.1).

The tools can read experiments created with the FCS version of Oracle Solaris Studio 12.2, as well as with the FCS and patched versions of Studio 12 Update 1, and Studio 12.

Experiments created with a version earlier than Sun Studio 12 cannot be read with Oracle Solaris Studio 12.2 tools.

Changes to Performance Analyzer Tool

The Performance Analyzer tool features the following enhancements.

New Call Tree Tab

The new Call Tree tab displays a dynamic call graph of the program as a tree with each function call shown as a node that you can expand and collapse. An expanded function node shows all the function calls made by the function, plus performance metrics for those function calls. When you select a node, the Summary tab on the right displays metrics for the function call and its callees. The percentages given for attributed metrics are the percentages of the total program metrics.

To easily find the branch that is consuming the most time, right click any node and select Expand Hottest Branch.

Enhancements to the Callers-Callees Tab

You can construct a call stack fragment in the center Stack Fragment panel, one call at a time, by adding callers and callees to the call stack. Callers are functions that call the fragment; callees are functions called from that fragment. Features include:

- As you add and remove functions in the stack fragment, the metrics are computed for the entire fragment and displayed next to the last function in the fragment.
- You can right-click on a caller to add a function to the top of the stack fragment, and right-click on a callee to add a function at the bottom. You can also use buttons above the Stack Fragment panel to manipulate the call stack fragment.
- You can use the Back and Forward buttons located above the Stack Fragment panel to go through the history of your changes to the call stack fragment.
- You can filter data in the Callers-Callees tab from the context (right-click) menu.

New Comparing Experiments Feature

Performance Analyzer now enables you to compare experiments that have been collected on the same executable. This feature is only partially implemented and might change in a subsequent release. In the current release, comparing experiments works as follows:

- If you open two or more experiments or experiment-groups, the data is aggregated by default.
- If you add `compare on` to your `.er.rc` file, and open two or more experiments or experiment-groups in the Performance Analyzer, the data is shown in a comparison mode.
- In comparison mode, the data from the experiments or groups is shown in adjacent columns with an additional header line that shows the experiment or group name. The columns are shaded in color to distinguish the experiments or groups.
- The tabs that support comparing experiments are Functions, Callers-Callees, Source, Disassembly, Lines, and PCs. You can disable and enable comparison mode from a context menu in the any of these tabs.
- You can also enable and disable comparison mode in Analyzer's Set Data Presentation dialog using the Compare Experiments option in the Formats tab.

Miscellaneous Enhancements

- Highlighting in the Source tab shows hot (highest CPU usage) lines in orange, and shows non-zero metric lines in yellow.
- A context menu in the Source tab allows navigation to the next or previous hot line, or non-zero metric line.
- You can create JPG files of the Timeline, the MPI Timeline, and the MPI Charts through the Print menu.
- Source and Dissasembly of HotSpot-compiled code exploits better mappings, where recorded.

The `er_print` Command

The `er_print` command is changed in this release as follows:

- New commands for controlling the Callers-Callees list now support call stack-building. The new `er_print` subcommands `cprepend`, `cappend`, `crmfirst`, and `crmlast` add or remove functions from the call stack fragment you are building. After each command, the caller-callee data for the current fragment is written.
- A new `calltree` command prints the dynamic call graph of the target showing the hierarchical metrics for all functions.
- A new `describe` command describes the recorded data from the experiments, and prints the tokens available for filtering.
- Source and Dissasembly of HotSpot-compiled code exploits better mappings, where recorded.
- The `er_print` command now enables you to compare experiments that have been collected on the same executable. This feature is only partially implemented and might change in a subsequent release. In the current release, comparing experiments works as follows:
  - When `er_print` is invoked on two or more experiments or experiment-groups, the data will be aggregated.
  - If you put `compare on` in your `.er.rc` file, and run `er_print` on two or more experiments or experiment-groups, the data is shown in a comparison mode.
  - In comparison mode, the data from the experiments or groups is shown in adjacent columns on the Functions list, the Caller-callees list, and the Source and Disassembly lists. The columns are shown in the order of the loading of the experiments or groups, with an additional header line giving the experiment or group name. Comparison mode is enabled and disabled with the `compare` command.

New Data Collection Features

The `collect` command is changed in this release as follows:

- The default setting for following descendants has been changed to `-F on`.
- MPI experiments with any release of Sun HPC ClusterTools, now known as Oracle Message Passing Toolkit, can be specified by `-M OMPT` or `-M CT`
- MPI experiments now also follow descendant processes by default.
- Handling of post-processing for MPI tracing experiments is improved.
- Support is added for hardware counter profiling on Oracle Enterprise Linux.
- Hardware counter aliases have been improved, and support is added for hardware counter profiling on the following processors:
  - SPARC64 VI and VII
  - Intel Core i7: Family 6, Models 30, 31, 37, 44, and 46 (including Nehalem EP and EX)
  - AMD Family 10h and 11h
- Experimental support for profiling scripts has been implemented, and may change in a subsequent release. To profile a script, set the environment variable `SP_COLLECTOR_SKIP_CHECKEXEC` and pass the script name to `collect`.
- Java profiling has been enhanced to provide more detailed information for source-line mappings for HotSpot-compiled code. The Java profiling enhancement is supported for JDK 1.6u20 or later JDK 1.6 updates, and from JDK 1.7.0-ea-b85 or later JDK 1.7 updates.
- The default size limit for experiments has been removed. You can use the `-L` option to set a size limit.

New dbx collector Features

The `collector` subcommand for the dbx debugger is changed as follows:

- Hardware counter aliases have been improved, and support is added for hardware counter profiling on the following processors:
  - SPARC64 VI and VII
  - Intel Core i7: Family 6, Models 30, 31, 37, 44, and 46 (including Nehalem EP and EX)
  - AMD Family 10h and 11h
- The default size limit for experiments has been removed. The `collector limit` command can be used to set a size limit.

A Change to `er_kernel`

The command for profiling a Solaris kernel is changed so that `er_kernel` will do the following when any of the signals `SIGINT`, `SIGTERM` or `SIGQUIT` are sent to the process:

- Catch `SIGINT`, `SIGTERM` or `SIGQUIT`
- Terminate the experiment
- Run `er_archive` if `-A off` is not specified

New command `er_generic`

The `er_generic` command generates an experiment from text files containing profile information. The simulated experiment can then be examined using the Performance Analyzer or `er_print` command. See the `er_generic`(1) man page for more information.

Change to `en_desc`

By default, the `en_desc` command now reads all descendants.

The Thread Analyzer

The Thread Analyzer now supports data race detection for code that is instrumented at either the source level or binary level. Source-level instrumentation is unchanged in this release.

To instrument a program's binary code, you need to use the `discover` tool, which is included in Oracle Solaris Studio and is documented in the `discover`(1) man page. See also *Oracle Solaris Studio 12.2 Discover and Uncover User's Guide*

For instrumenting a program's binary code to detect data races, the `discover` tool requires the input binary to be compiled under the following conditions:

- Operating system version must be at least Oracle Solaris 10 5/08 or OpenSolaris version snv_70
- Compiler must be from a release no earlier than Sun Solaris Studio 12 Update 1
- One of the compiler optimization flags `(-xO1, -xO2, -xO3, -xO4, -xO5)` must be used

You might also be able to use the `discover` tool on an earlier Solaris version running on a SPARC-based system if the binary was compiled with the compiler option `-xbinopt=prepare`. See the `cc`(1), `CC`(1), or `f95`(1) man pages for information about this compiler option.

If the binary is named `a.out`, you can create an instrumented binary named `a.out_i` with the following command:

```
% discover -i datarace -o a.out_i a.out
```

See the *Oracle Solaris Studio 12.2: Thread Analyzer User's Guide* or `tha`(1) man page for more information.

CHAPTER 5

Debugging Tools

What's new in the debugging tools in this Oracle Solaris Studio release.

dbx

New and Changed Features

The following features were added or changed in Oracle Solaris Studio 12.2 dbx.

- Improved support for optimized code debugging:
  - Information for locating parameters and local variables is available on x86 platforms.
  - Information on inline functions is available for SPARC platforms.
- New commands for displaying information on OpenMP regions, tasks, and thread sets:
  - `omp_pr` [*parallel_region_id*] [`-ancestors`|`-tree`] [`-v`]

    Print a description of the current parallel region or the region specified by the *parallel_region_id*, including the parallel region id, type (implicit or explicit), state (active or inactive), team size (number of threads), and program location (program counter address).
  - `omp_tr` [*task_region_id*] [`-ancestors`|`-tree`]

    Print a description of the current task region or the region specified by the *task_region_id*, including the task region id, type (implicit or explicit, tied or untied), state (spawned, executing, or waiting), encountering thread, executing thread, program location, unfinished children, and parent.
  - `omp_team` [*parallel_region_id*]

    Print all the threads in the current team. If *parallel_region_id* is specified, print the threads in the team for that region.
  - `omp_loop`

    Print a description of the current loop, including scheduling type (static or dynamic), wait or nowait, ordered, bounds, and number of iterations. This command can be issued only from the thread that is currently executing the loop.

  - `omp_serialize`

    Serialize the next encountered parallel region for the current thread.

- Extensions of existing commands for OpenMP programs:
  - `print -s` *expression*
  - `thread -info`
  - `what is` *name*
  - `where`
- New OpenMP events:
  - `omp_barrier` [*type*] [*state*]

    Tracks the event of a thread entering a barrier.

  - `omp_taskwait` [*state*]

    Tracks the event of a thread entering a taskwait.

  - `omp_ordered` [*state*]

    Tracks the event of a thread entering an ordered region.

  - `omp_critical`

    Tracks the event of a thread entering a critical region.

  - `omp_atomic` [*state*]

    Tracks the event of a thread entering an atomic region.

  - `omp_flush` [*type*]

    Tracks the event of a thread executing a flush.

  - `omp_task` [*state*]

    Tracks the creation and termination of tasks.

  - `omp_master`

    Tracks the event of a master thread entering the master region.

  - `omp_single`

    Tracks the event of a thread entering a single region.

Software Corrections

This section describes problems fixed in this release of Oracle Solaris Studio 12.2 dbx

1. Can't stop dbx execution when doing tracei step

The user is unable to stop dbx by typing control-C (^C) when doing a `tracei` step on a Solaris platform. This is actually a Solaris OS bug, but dbx was modified to work around the problem.

2. Unable to step into a instrumented debuglog uttsc binary

   dbx did not correctly handle a C++ namespace alias in a lexical block. This created a problem where dbx could not step into a specially instrumented binary.
3. Threads related commands not available in dbx for multithreaded program instrumented with Purify

   Purify adds suffixes to the names of all the shared libraries that it instruments. For example, `libc.so.1` becomes `libc.so.1_pure_p3_c0_1005282029_510_32`. dbx made decisions based on the presence of `libc.so.1`, which was no longer being loaded. dbx now understands the `_pure*` suffix.
4. dbx unable to demangle certain GCC 4.x. sybx is unable to extract the program name from a core file on SLES 10.2

   On an SuSE Linux Enterprise Server 10.2 system, dbx was unable to extract the program from a core file, because newer Linux systems include two note sections in a core file, and the second one is empty. dbx will get the name from the first section.
5. dbx finds copies of a constructor in gcc code

   gcc sometimes generated multiple entries for a member, which included one in DWARF debug.pubnames section, one as prototype, and an abstraction instance. dbx needed to delete the prototype entry once its instance is seen and processed.
6. dbx is unable to extract program name from a core file on SLES 10.2

   On an SuSE Linux Enterprise Server 10.2 system, dbx was unable to extract the program from a core file, because newer Linux systems include two note sections in a core file, and the second one is empty. dbx will get the name from the first section.
7. dbx gets SIGSEGV on printing a variable

   When the executable is built from some object files (.o) that were not compiled with the `-g` option, and dbx needs to import one of these to evaluate the variable, the import could fail because dbx was not checking for this condition.
8. dbx — core segv if is not valid

   dbx did not check for the the possibility of a zero length core file and did not handle it gracefully.
9. Memory and Disassembler windows can cause dbx to crash under IDE

   If a debug session in the IDE was ended while displaying the Memory or Disassembler windows, and then the session was restarted, fronting either of these windows would crash the underlying dbx.

dbxtool

The following are new and changed features in dbxtool. (See the `dbxtool(1)` man page and the Oracle Solaris Studio 12.2 dbxtool Tutorialfor details.)

- The global Debugging Options in the Options window have been rearranged into Session Startup properties and Window properties. Four property settings have been eliminated: Front The Dbx Commands Tab When The Program Stops, Save and Restore Breakpoints, Allow Step To Start Process, and Balloon Expression Evaluation.
- The Local Variables window is now the Variables window .
- The New Watch button and the Expression Evaluation button have been removed from the toolbar.
- In the New Breakpoint dialog box: the LWP, language mode, and temporary checkboxes have been removed. The Condition, Count, WhileIn, and Thread fields have been reordered. The More/Less button has been removed.
- The Sessions window opens automatically if you have more than one debugging session.
- The Call Stack window shows up to 40 frames; user can click More to see more than 40.
- The Variables window and Watch window can show static members.
- The Variables window has a button for displaying autos only.
- The Variables window has a New Watch button.
- The Disassembler window has been renamed Disassembly.

Discover and Uncover

The Sun Memory Error Discovery Tool (Discover), an advanced development tool for detecting memory access errors, is new in this release.

Uncover, a simple and easy to use command-line tool for measuring code coverage of applications, is new in this release.

See the `discover` and `uncover(1)` man pages, and the *Oracle Solaris Studio 12.2 Discover and Uncover User's Guide* for details.

DLight

DLight, a stand-alone interactive graphical observability tool for C/C++ developers based on Oracle Solaris Dynamic Tracing (DTrace) technology, is new in this release. This is not the same DLight tool that was included in Sun Studio 12 Update 1. See the Oracle Solaris Studio 12.2 DLight Tutorial

CHAPTER 6

The Solaris Studio IDE

The Oracle Solaris Studio 12.2 IDE (Indegrated Development Environment) provides modules for creating, editing, building, debugging, and analyzing the performance of a C, C++, or Fortran application. This chapter highlights important information about the IDE in this Oracle Solaris Studio release.

The command to start the IDE is `solstudio`. For details on this command, see the `solstudio`(1) man page.

For complete documentation of the IDE, see the online help in the IDE and the Oracle Solaris Studio 12.2 IDE Quick Start Tutorial

New and Changed Features

The following features were added or changed in the Oracle Solaris Studio 12.2 IDE:

- Based on NetBeans IDE 6.9
- The Qt application development framework lets you create Qt files, such as GUI forms, resources, and translations.
- The Run Monitor displays information about the application runtime such as CPU, memory, and thread usage. On Solaris platforms you can track Thread Microstates with Thread Details, and I/O usage.
- The Call Graph now includes a graphical view as well as a tree view of all of the functions called from a selected function or all the functions that call that function.
- Hyperlink navigation now lets you jump from a method that is overridden to the method that overrides it, and the reverse
- You can add comments to your source code to generate documentation for your functions, classes, and methods. The IDE recognizes comments that use Doxygen syntax and automatically generates documentation.

- The global Debugging Options in the Options window have been rearranged into Session Startup properties and Window properties. Four property settings eliminated: Front The Dbx Commands Tab When The Program Stops, Save and Restore Breakpoints, Allow Step To Start Process, and Balloon Expression Evaluation.
- The Local Variables window is now the Variables window.
- The New Watch button and the Expression Evaluation button have been removed from the debug toolbar.
- The Make Caller Current and Make Callee Current buttons have been removed from the debug toolbar.
- The Restart button has been added to the toolbar.
- In the New Breakpoint dialog box: the LWP, language mode, and temporary checkboxes have been removed. The Condition, Count, WhileIn, and Thread fields have been reordered. The More/Less button has been removed.
- The Sessions window opens automatically if you have more than one debugging session
- The Call Stack window shows up to 40 frames; user can click More to see more than 40.
- The Variables window and Watch window can show static members.
- The Variables window has a button for displaying variables only for the current source code line and the previous source code line.
- The Variables window has a New Watch button
- The Disassembler window has been renamed to Disassembly.

Software Requirements

The Oracle Solaris Studio IDE requires the Java SE Development Kit (JDK) 6 Update 13 or later. If the IDE cannot find the required JDK, it does not start and issues an error message.

Updating the IDE

The IDE's Plugins manager enables you to update your IDE's installed plugins dynamically. You can also use the Plugins manager to add new plugins and functionality to the IDE.

When you use the Plugins manager to update the IDE, the IDE checks the registered update centers to see if there are new plugins or new versions of already installed plugins available. If new or updated plugins are available, you can select, download, and install the plugins using the Plugins manager.

Alternatively, you can choose Help > Check For Updates to open the Plugin Installer. The Plugin Installer will check for updates of installed plugins. If updates are available, you can step through the installer to install the updates.

In addition to the default IDE Update Center, you can choose from several update centers that offer different types of plugins, such as experimental new plugins or old plugins that are no longer in regular distribution.

To update installed plugins from the Update Center:

1. Choose Tools > Plugins to open the Plugins manager.
2. Click the Updates tab to display available updates of installed plugins.
3. In the left pane, select the plugins you wish to update and click Update.
4. Complete the pages in the installer to download and install the update.

The left pane of the Updates tab displays the installed plugins that have updates available from the update centers. By default, the IDE regularly checks the registered update centers for available updates of installed plugins. If no plugins are displayed in the left pane, it means that no updates were available the last time the IDE checked the update center.

To add new plugins from the Update Center:

1. Choose Tools > Plugins to open the Plugins manager.
2. Click the Available Plugins tab to display plugins that are available but not installed.
3. In the left pane, select the plugins you wish to add and click Install.
4. Complete the pages in the installer to download and install the plugin.

Some plugins may require you to restart the IDE to complete the update process.

You can set the frequency that the IDE checks for updates in the Settings tab of the Plugins manager. You can click Reload Catalog to check the update centers immediately.

Configuration

NetBeans IDE 6.9 has a default heap size of 128 MB. The Oracle Solaris Studio 12.2 IDE runs well with this default setting when you are developing small projects with up to 500 source and header files.

When you are developing larger projects, you will need to increase the heap size. If you get an OutOfMemory exception when developing a large project, the heap size is a likely cause.

You can set the heap size for the Java Virtual Machine (JVM)* on which the NetBeans IDE runs in the netbeans.conf file.

To change the heap size:

- In the /Oracle_Solaris_Studio_installation_directory/netbeans/etc/netbeans.conf file, edit the –`J-Xmx` command line Java startup switch (bolded below) in the netbeans.conf file, and then restart the IDE.

  ```
  netbeans_default_options="-J-Xms32m –J-Xmx128m –J-XX:PermSize=32m
  –J-XX:MaxPermSize=96m –J-Xverify:none –J-Dapple.laf.useScreenMenuBar=true"
  ```

The recommended heap sizes for NetBeans C/C++ Plugin for medium and large applications are:

- For developing medium applications (500–2000 source and header files) on a system with 1 GB or more of RAM: 512 MB
- * For developing large applications (more than 2000 source and header files) on a system with 2 GB or more of RAM): 1 GB

If you are running the Sun JVM, you can also add the garbage collector switches —J-XX:+UseConcMarkSweepGC (concurrent collector) and —J-XX:+UseParNewGC (parallel collector) to the netbeans.conf file. These options allow the garbage collector to run in parallel with the main execution engine. They might not be supported by non-Sun implementations of the JVM.

For more information on NetBeans performance tuning, see Tuning JVM Switches for Performance.

Note: The terms “Java Virtual Machine” and “JVM” mean a Virtual Machine for the Java(TM) platform.

CHAPTER 7

Other Tools

What's new in dmake and the software installer in this Oracle Solaris Studio release.

Dmake

dmake is a command-line tool, compatible with make(1). dmake can build targets in grid, distributed, parallel, or serial mode. If you use the standard make(1) utility, the transition to dmake requires little if any alteration to your makefiles. dmake is a superset of the make utility. With nested makes, if a top-level makefile calls make, you need to use $(MAKE). dmake parses the makefiles and determines which targets can be built concurrently and distributes the build of those targets over a number of hosts set by you.

dmake is integrated with the Solaris Studio IDE. By default all projects are built with dmake, which runs in parallel mode. Project properties let users specify the maximum number of build jobs. By default dmake runs 2 jobs in parallel, which means many projects will build twice as fast on multi-CPU systems.

For information about how to use dmake, see the *Distributed Make (dmake)* manual.

Software Corrections In This Release

- Fixed bug: dmake dumps core when handling conditional macros with long contents
- Fixed bug: DMAKE_OUTPUT_MODE values differ between implementation (TXT1/TXT2) and docs (TEXT1/TEXT2). Now dmake accepts values "TEXT1" and "TEXT2" as well.
- Fixed bug: 'dmake -v' prints wrong version on Linux Now dmake prints correct version.
- Fixed bug: Modula considered Harmful Old rules for Modula compiler are removed from make.rules file.
- Fixed bug: dmake memory leaks in the KEEP_STATE mode

Features Added in Previous Releases:

- dmake is now integrated in Solaris Studio IDE. This means that by default all projects are built using dmake in parallel mode. To change the build mode or change the number of parallel jobs can be done from inside the IDE:
  1. From the main menu, choose Tools -> Options to open the "Options" dialog
  2. In "Options", select the C/C++ icon (left panel) to show the C/C++ options on right panel
  3. Click on the "Project Options" tab (right panel) to show the project options, and select "Make Options"
  4. Enter **`-m parallel -j 24`**
  5. Press "Ok" button.

  Now all projects will be built in parallel mode up to a maximum of 24 jobs.
- The `-x SUN_MAKE_COMPAT_MODE=`*compatibility-mode* command-line option:

  `-x SUN_MAKE_COMPAT_MODE=SUN` (default) for compatibility with Solaris `make`

  `-x SUN_MAKE_COMPAT_MODE=POSIX` for compatibility with POSIX `make`

  `-x SUN_MAKE_COMPAT_MODE=GNU` for compatibility with GNU `make`
- Similarly, the `SUN_MAKE_COMPAT_MODE` environment variable gives users the same three options to specify dmake's behavior in compatibility mode:

  `SUN_MAKE_COMPAT_MODE=SUN` (default) for compatibility with Solaris `make`

  `SUN_MAKE_COMPAT_MODE=POSIX` for compatibility with POSIX `make`

  `SUN_MAKE_COMPAT_MODE=GNU` for compatibility with GNU `make`
- UNIX 2003 compliance. `dmake` and the `make` utility in the Solaris 10 OS passed the UNIX 2003 conformance tests (XPG5)
- `dmake` now includes support for the Sun Grid Engine on the AMD64 architecture.
- System overloading control is now available on the AMD64 architecture.
- The `DMAKE_OUTPUT_MODE` environment variable gives you two format options for the log file, one of which serializes the output of parallel jobs, making the log file more readable.

Solaris Studio Installer

The new and changed features in the Installer include:

- The list of components you can choose to install has changed. The selectable components now are:
  - Compiler support files (required by the C and C++ compilers)
  - C and C++ compilers

  - Fortran compiler
  - dbx debugger
  - dbxtool
  - dmake
  - DLight observability tool (Solaris platforms only)
  - IDE
  - Performance and Thread Analysis tools (Solaris platforms only)
  - Performance Library (Solaris platforms only)
  - ScaLAPACK (Solaris platforms only)
- You can now install only the runtime libraries by starting either the GUI installer or the non-GUI installer with the –libraries-only option. This option installs the C++ libraries, the Fortran 90 libraries, the math libraries (Solaris platforms only).
- If you are installing on a system with zones, you can now specify installation in the current zone only using a checkbox in the GUI installer, as well as from the command line when starting the non-GUI installer.
- Alternative root installation is now supported only in the non-GUI installer, and not in the GUI installer.
- The installers do not require you to accept a license.
- There is no longer a restriction on the zone in which you can run the installer if you have previous Sun Studio releases installed on the system.

CHAPTER 8

Known Problems, Limitations, and Workarounds in This Release

Here are some of the known issues at the time of this release, and information about how to work around these problems.

Compilers

This section describes known issues, problems, and workarounds for the compilers in this release.

Issues Common To The Compilers

Known problems related to —xprofile

- `libxprof` fails if different versions of same object file loaded in same process

  This can occur if two different versions of the same file are built with the same name in the same directory, linked into two different shared libraries, and loaded into the same process, perhaps not simultaneously.

  Workaround: When building shared libraries with `-xprofile`, ensure that object files names have distinct UNIX pathnames. Note that pathnames can still be distinct even if their basenames are not. For example,

  ```
  /work/mylib/unshared/x.o
  /work/mylib/shared/x.o
  ```

  are considered distinct.
- OMP: `libxprof`: Assertion failed

  An assertion failure may occur under low memory conditions if `malloc()` fails during a call to a profiling runtime routine.

  Workaround: Add memory or swap space.

- `-xprofile=tcov:`*prof_dir* resolves relative *prof_dir* incorrectly

  Under –`xprofile=tcov:`*dir*, a non-absolute UNIX pathname is resolved relative to the directory in which object file will be generated.

  Workaround: Use absolute pathnames with –`xprofile={collect,use,tcov}:`*dir*.

C++

Correct Interpretation of Large Decimal Integer Constants

The C++ standard says that a decimal integer constant without a suffix is treated as an `int` if the value fits in an `int`, otherwise as a `long int`. If the value does not fit in a `long int`, the results are undefined.

In 32-bit mode, types `int` and `long` have the same size and data range. The C++ compiler followed the 1990 C standard rule, treating a value between `INT_MAX+1` and `LONG_MAX` as unsigned `long`. This treatment produced unexpected results in some programs.

The 1999 C standard changed the rule about unsuffixed decimal integers so that they are never treated as unsigned types. The type is the first of `int`, `long`, or `long long` that can represent the value.

The C++ compiler follows the C99 rule when in standard mode, but continues to follow the C90 rule in `-compat=4` mode. (In `-compat=4` mode, the compiler behaves like the C++ 4.2 compiler.)

If you want a large decimal integer to be treated as unsigned, the portable solution is to use a `u` or `U` suffix. You can use the other suffixes for other types as well. For example:

```
// note: 2147483648 == (INT_MAX+1)
2147483648     // (signed) long long
2147483648LL   // (signed) long long
2147483648U    // same as 2147483648u
```

Ambiguity: Constructor Call or Pointer-to-Function

Some C++ statements could potentially be interpreted as a declaration or as an expression-statement. The C++ disambiguation rule is that if a statement can be a declaration, it is a declaration.

Earlier versions of the compiler misinterpreted cases like the following:

```
struct S {
  S();
};
struct T {
  T( const S& );
};
T v( S() );    // ???
```

The programmer probably intended the last line to define a variable `v` initialized with a temporary of type `S`. Previous versions of the compiler interpreted the statement that way.

But the construct "`S()`" in a declaration context can also be an abstract declarator (that is, one without an identifier) meaning "function with no parameters returning of value of type `S`." In that case, it is automatically converted to the function pointer "`S(*)()`". The statement is thus also valid as a declaration of a function v having a parameter of function-pointer type, returning a value of type `T`.

The compiler now makes the correct interpretation, which might not be what the programmer intended.

There are two ways to modify the code to make it unambiguous:

```
T v1( (S()) );  // v1 is an initialized object
T v2( S(*)() ); // v2 is a function
```

The extra pair of parentheses in the first line is not valid syntax for `v1` as a function declaration, so the only possible meaning is "an object of type `T` initialized with a temporary value of type `S`."

Similarly, the construct "`S(*)()`" cannot possibly be a value, so the only possible meaning is as a function declaration.

The first line can also be written as

```
T v1 = S();
```

Although the meaning is completely clear, this form of initialization can sometimes result in extra temporaries being created, although it usually does not.

Writing code like the following is not recommended because the meaning is not clear, and different compilers might give different results.

```
T v( S() ); // not recommended
```

Template Syntax Error is No Longer Ignored

The following template syntax has never been valid, but Sun C++ 4 and 5.0 did not report the error. All versions since 5.1 of the C++ compiler report this syntax error when you compile in standard mode (the default mode).

```
template<class T> class MyClass<T> { ... }; // definition error
template<class T> class MyClass<T>; // declaration error
```

In both cases the <T> in `MyClass<T>` is invalid and must be removed, as shown below:

```
template<class T> class MyClass { ... }; // definition
template<class T> class MyClass; // declaration
```

Linking Fails If You Combine -xipo or -xcrossfile With -instances=static

The template option `-instances=static` (or `-pto`) does not work in combination with either of the `-xcrossfile` or `-xipo` options. Programs using the combination will often fail to link.

If you use the `-xcrossfile` or `-xipo` options, use the default template compilation model, `-instances=global` instead.

In general, do not use `-instances=static` (or `-pto`) at all. It no longer has any advantages, and still has the disadvantages documented in the C++ Users Guide.

Cross-Language Linking Error

The `-xlang=f77` command-line option causes the compilation process to encounter a linker error. To avoid the error and still include the appropriate runtime libraries, compile with`-xlang=f77,f90` instead.

Name Mangling Linking Problems

The following conditions may cause linking problems.

- A function is declared in one place as having a `const` parameter and in another place as having a non-`const` parameter.

  Example:

  ```
  void foo1(const int);
  void foo1(int);
  ```

  These declarations are equivalent, but the compiler mangles the names differently. To prevent this problem, do not declare value parameters as `const`. For example, use `void foo1(int);` everywhere, including the body of the function definition.
- A function has two parameters with the same composite type, and just one of the parameters is declared using a typedef.

  Example:

  ```
  class T;
  typedef T x;
  // foo2 has composite (that is, pointer or array)
  // parameter types
  void foo2(T*, T*);
  void foo2(T*, x*);
  void foo2(x*, T*);
  void foo2(x*, x*);
  ```

  All declarations of `foo2` are equivalent and should mangle the same. However, the compiler mangles some of them differently. To prevent this problem, use typedefs consistently.

  If you cannot use typedefs consistently, a workaround is to use a weak symbol in the file that defines the function to equate a declaration with its definition. For example:

  ```
  #pragma weak "__1_undefined_name" = "__1_defined_name"
  ```

Note that some mangled names are dependent on the target architecture. (For example, `size_t` is `unsigned long` for the SPARC V9 architecture (`-m64`), and unsigned `int` otherwise.) In such a case, two versions of the mangled name are involved, one for each model. Two pragmas must be provided, controlled by appropriate `#if` directives.

Debugging Tools Erroneously Report Extra Leading Parameter for Member Functions

In compatibility mode (`-compat`), the C++ compiler incorrectly mangles the link names for pointers to member functions. The consequence of this error is that the demangler, and hence some debugging tools, like dbx and c++filt, report the member functions as having an extra leading parameter consisting of a reference to the class type of which the function is a member. To correct this problem, add the flag `-Qoption ccfe -abiopt=pmfun1`. Note, however, that sources compiled with this flag may be binary incompatible with sources compiled without the flag. In standard mode (the default mode), the problem does not occur.

No Support For Referencing a Non-Global Namespace Object From a Template

A program using templates and static objects causes link-time errors of undefined symbols if you compile with `-instances=extern`. This is not a problem with the default setting `-instances=global`. The compiler does not support references to non-global namespace-scope objects from templates. Consider the following example:

```
static int k;
template<class T> class C {
        T foo(T t) { ... k ... }
};
```

In this example, a member of a template class references a static namesapce-scope variable. Keep in mind that namespace scope includes file scope. The compiler does not support a member of a template class referencing a static namespace-scope variable. In addition, if the template is instantiated from different compilation units, each instance refers to a different k, which means that the C++ One-Definition Rule is violated and the code has undefined behavior.

Depending on how you want to use k and the effect it should have, the following alternatives are possible. The second option only is available for function templates that are class members.

1. You can give the variable external linkage:

   ```
   int k; // not static
   ```

   All instances use the same copy of `k`.

2. You can make the variable a static member of the class:

   ```
   template<class T> class C {
           static int k;
   ```

```
                T foo(T t) { ... k ... }
        };
```

Static class members have external linkage. Each instance of `C<T>::foo` uses a different `k`. An instance of `C<T>::k` can be shared by other functions. This option is probably what you want.

#pragma align Inside Namespace Requires Mangled Names

When you use `#pragma align` inside a namespace, you must use mangled names. For example, in the following code, the `#pragma align` statement has no effect. To correct the problem, replace `a,` `b`, and `c` in the `#pragma align` statement with their mangled names.

```
namespace foo {
  #pragma align 8 (a, b, c) // has no effect
  //use mangled names: #pragma align 8 (__1cDfooBa_, __1cDfooBb_, __1cDfooBc_)
  static char a;
  static char b;
  static char c;
}
```

Function Overload Resolution

Early releases of the C++ compiler did not perform function overload resolution in accordance with the requirements of the C++ standard. The current release fixes many bugs in resolving calls to overloaded functions. In particular, the compiler would sometimes pick a function when a call was actually ambiguous, or complain that a call was ambiguous when it actually was not.

Some workarounds for ambiguity messages are no longer necessary. You may see new ambiguity errors that were not previously reported.

One major cause of ambiguous function calls is overloading on only a subset of built-in types.

```
int f1(short);
int f1(float);
...
f1(1); // ambiguous, "1" is type int
f1(1.0); // ambiguous, "1.0" is type double
```

To fix this problem, either do not overload f1 at all, or overload on each of the types that do not undergo promotion: int, unsigned int, long, unsigned long, and double. (And possibly also types long long, unsigned long long, and long double.)

Another major cause of ambiguity is type conversion functions in classes, especially when you also have overloaded operators or constructors.

```
class T {
public:
        operator int();
```

```
                T(int);
                T operator+(const T&);
        };
        T t;
        1 + t // ambiguous
```

The operation is ambiguous because it can be resolved as

```
         T(1) + t     // overloaded operator
        1 + t.operator int()    // built-in int addition
```

You can provide overloaded operators or type conversion functions, but providing both leads to ambiguities.

In fact, type conversion functions by themselves often lead to ambiguities and conversions where you did not intend for them to occur. If you need to have the conversion available, prefer to use a named function instead of a type conversion function. For example, use `int to_int();` instead of operator `int();`

With this change, the operation `1 + t` is not ambiguous. It can be interpreted only as `T(1) + t`. If you want the other interpretation, you must write `1 + t.to_int()`.

Fortran

The following issues should be noted in this release of the `f95` compiler:

- Removal in this release of the obsolete FORTRAN 77 libraries means that old executables compiled with the legacy Sun WorkShop `f77` compiler that depend on the shared libraries `libF77`, `libM77` and `libFposix` will not run.
- Using an enhanced array constructor to set up a parameter constant whose type is assumed-length character will result in the values of the character elements concatenated together. The workaround is to use the same character length as used in the array constructor to define the parameter constants instead of using assumed-length.
- Specifying a C binding procedure with a blank name is mishandled and causes a blank name to be used for the procedure. The workaround is to specify a C binding name or not use a C binding name if it is not needed.

Previous releases of the f95 compiler introduced incompatibilities that carry forward to this release of the compiler and should be noted if you are updating from earlier f95 releases. The following incompatibilies are worth noting:

Array Intrinsic Functions Use Global Registers:

The array intrinsic functions `ANY, ALL, COUNT, MAXVAL, MINVAL, SUM, PRODUCT, DOT_PRODUCT`, and `MATMUL` are highly tuned for the appropriate SPARC platform architectures. As a result, they use the global registers `%g2, %g3`, and `%g4` as scratch registers.

User code should not assume these registers are available for temporary storage if calls are made to the array intrinsics listed above. Data in these registers will be overwritten when the array intrinsics are called.

F95 Modules in Archive Libraries Not Included In Executable:

The debugger dbx requires all object files used in the compilation to be included in the executable file. Usually, programs satisfy this requirement with no extra work on the part of the user. An exceptional case arises from the use of archives containing modules. If a program uses a module, but does not reference any of the procedures or variables in the module, the resulting object file will not contain references to the symbols defined in the module. The linker only links with a object file from an archive if there is a reference to a symbol defined in the object file. If there is no such reference, the object file will not be included in the executable file. Dbx will give a warning when it tries to find the debugging information associated with the module that was used. It will not be able to provide information about the symbols whose debugging information is missing.

Use the `-u` linker option to work around this problem. This option takes a symbol as its option argument. It adds that symbol to the set of undefined linker symbols, so it will have to be resolved. The linker symbol associated with a module is normally the module name with all letters in lower case followed by an underscore.

For example, to force the object file containing the module `MODULE_1` to be taken from an archive, specify the linker option `-u module_1_`. If linking using the f95 command, use `-Qoption ld -umodule_1_` on the command line.

Tools

dbx

Known dbx issues and Workarounds

1. Data Collection Problems When dbx is Attached to a Process

   If you attach dbx to a running process without preloading the collector library, libcollector.so, a number of errors can occur.

   - You cannot collect any tracing data: synchronization wait tracing, heap tracing, or MPI tracing. Tracing data is collected by interposing on various libraries, and if libcollector.so is not preloaded, the interposition cannot be done.
   - If the program installs a signal handler after dbx is attached to the process, and the signal handler does not pass on the SIGPROF and SIGEMT signals, profiling data and sampling data is lost.

  - If the program uses the asynchronous I/O library, `libaio.so`, clock-based profiling data and sampling data is lost, because `libaio.so` uses SIGPROF for asynchronous cancel operations.
  - If the program uses the hardware counter library, `libcpc.so`, hardware-counter overflow profiling experiments are corrupted because both the collector and the program are using the library. If the hardware counter library is loaded after dbx is attached to the process, the hardware-counter experiment can succeed provided references to the `libcpc` library functions are resolved by a general search rather than a search in `libcpc.so`.
  - If the program calls `setitimer`(2), clock-based profiling experiments can be corrupted because both the collector and the program are using the timer.

2. dbx might crash while debugging Java code

   If you issue a **`cd`** command from within the dbx shell, or set the `CLASSPATH` environment variable or the `CLASSPATHX` environment variable, dbx might subsequently crash with a segmentation fault.

   Workarounds:

   - Do not do any of the above.
   - Delete all watches (displays) before doing any of the above.

3. dbx crashes on re-debugging of Java code

   Issuing two debug commands in a row on Java code might cause dbx to crash.

4. dbx throws an exception when debugging application on different J2SE than it was built on

   dbx throws an exception when you debug an application under a different release of the J2SE technology than the version of the J2SE technology under which you built the application.

5. False RUA error reported due to pre-RTC monitoring allocations

   Under unusual circumstances with multithreaded programs, runtime checking (RTC) reports a false RUA error when it detects access to internal thread-related data that were allocated before RTC began monitoring memory allocations. As these circumstances are part of normal thread switching behavior, these false RUA reports can safely be ignored by using the dbx suppress command.

dbx Limitations and Incompatibilities

Oracle Solaris Studio 12.2 dbx has the following limitations:

- The following features of dbx are not available on the Linux OS on x86 based systems:
  - Fix and continue

    Performance data collection

    Breakpoints on the following events:

    - `fault`

```
lastrites
lwp_exit
sysin
sysout
sync
throw
```

- The following features of dbx are not available on the Linux OS on x64 based systems:
  - Java debugging
  - Debugging 32–bit programs, unless you start dbx with the `-x exec32` option.
- dbx cannot follow forked processes on Linux platforms or change to a new program when `exec()` is called
- The pipe operator in the Korn shell is limited on Linux platforms. Any dbx command that needs to access the target process does not work as part of a pipeline. For example, the following command is likely to cause dbx to hang:

  `where | head –1`

  Workarounds:
  - Type Ctrl-C to display a new dbx prompt.
  - dbx caches a lot of information, so for the above example, the following sequence of commands works:

    ```
    where
    where | head –1
    ```
- The following problems may occur when debugging programs on Linux platforms:
  - If a program uses `clone()` to implement its own style of threads, then thread support in dbx does not identify threads correctly.

    Workaround:

    Use `libthread.so` rather than `clone()`.
  - The threads library in the Linux OS uses SIGSTOP signals as part of its internal mechanism. Normally dbx hides these signals from you, and allows you to monitor genuine SIGSTOP signals from other sources. Occasionally the Linux OS uses SIGSTOP in an unexpected way and dbx interprets a system-generated SIGSTOP as a user-generated SIGSTOP.

    Workaround:

    Use the `ignore` command to tell dbx not to catch SIGSTOP signals.
  - Sometimes a thread exits, but the Linux OS does not report the exit to dbx. This happens less often when using the new threads library (NPTL).

When a thread exits and the exit is not reported, dbx waits for an event that will never happen and does not display a new prompt. This situation is most likely to occur after you have given a cont command in dbx, but it can also happen after the step up command, step command, next command, and other commands.

Workarounds:

- Sometimes typing Ctrl-C causes dbx to stop waiting and display a new prompt.
- If Ctrl-C does not work, exit dbx and restart it.

- Run-time type information for C++ expressions is not available for programs compiled with the g++ compiler.
- It is not possible to attach to a running process from your `.dbxrc` file. A `.dbxrc` file should not contain commands that execute your code. However, you can put such commands in a file, and then use the dbx `source` command to execute the commands in that file.
- dbx incorrectly demangles pointer to member functions for compat=4. This is not a problem for compat=5.

  Workaround: Recompile your program with:

  ```
  CC –compat=4 –Qoption ccfe –abiopt=pmfun1
  ```

  This flag introduces an ABI change and should not be used in production builds.
- On SPARC V9 (`-m64`) systems, use of the `call` command or printing function calls is not working with nested small structure as an argument or as a return value.
- Using older copies of `libC.so.5` or `libC.so.4` may cause problems for dbx in the area of C++ exceptions. Warning messages about bad stabs and unhandled exceptions may result.

  Workaround: Install the latest `libC.so.5` on all systems.
- Fortran users should compile with the `-stackvar` option to take full advantage of runtime checking.
- Some programs may not work properly with `-stackvar`. In such cases, try the `-C` compiler option, which will turn on array subscript checking without RTC.
- Follow fork may be unreliable for multithreaded applications.
- Use of the call command or printing function calls may cause deadlock situations with multithreaded applications.
- Do not use the fix and continue feature of dbx to change a header file if the file was part of a pre-compiled header (PCH) collection.
- The dbx command line interpreter is an older version of the Korn shell (ksh) that does not support Code Set Independence (CSI). Multi-byte characters can be misinterpreted when typed on the dbx command line.

Performance Analyzer

If your are using Linux with Oracle Message Passing Toolkit 8.2 or 8.2.1, you might need a workaround. The workaround is not needed for version 8.1 or 8.2.1c, or for any version if you are using an Oracle Solaris Studio compiler.

The Oracle Message Passing Toolkit version number is indicated by the installation path such as `/opt/SUNWhpc/HPC8.2.1`, or you can type `mpirun` –`V` to see output as follows where the version is shown in italics:

```
mpirun (Open MPI) 1.3.4r22104-ct8.2.1-b09d-r70
```

If your application is compiled with a GNU or Intel compiler, and you are using Oracle Message Passing Toolkit 8.2 or 8.2.1 for MPI, to obtain MPI state data you must use the `-WI` and `--enable-new-dtags` options with the Oracle Message Passing Toolkit `link` command. These options cause the executable to define `RUNPATH` in addition to `RPATH`, allowing the MPI State libraries to be enabled with the `LD_LIBRARY_PATH` environment variable.

dmake

This section discusses known `dmake` software problems and possible workarounds for those problems.

If there are any problems with using `dmake` in distributed mode, verify the following:

1. The `$HOME` environment variable is set to an accessible directory.

   ```
   % ls -la $HOME
   ```
2. The file `$HOME/.dmakerc` exists, is readable, and contains correct information.

   ```
   % cat $HOME/.dmakerc
   ```
3. All hosts mentioned in the `$HOME/.dmakerc` file are alive by using the `/usr/sbin/ping` command to check each host.

   ```
   % /usr/sbin/ping $HOST
   ```

   where `$HOST` is the name of the system, which is listed as the host in `$HOME/.dmakerc` file.
4. Verify that the path to the `dmake` binaries is correct by using the `dmake`, `rxm`, and `rxs` commands:

   ```
       % which dmake
       % which rxm
       % which rxs
   ```
5. The remote login (`rsh`) on each host works without a password, and each remote login takes an acceptable time (less than 2 seconds).

   ```
   % time rsh $HOST uname -a
   ```

6. The file /etc/opt/SPROdmake/dmake.conf exists on each host and contains the correct information. If this file does not exist, dmake will distribute only one job on this system:

   ```
   % rsh $HOST cat /etc/opt/SPROdmake/dmake.conf
   ```

7. The path to the dmake binaries is correct for each host:

   ```
   % rsh $HOST ‘which dmake‘
   % rsh $HOST ‘which rxm‘
   % rsh $HOST ‘which rxs‘
   ```

8. The build area is available from each host (rwx):

   ```
   % cd $BUILD
   % rm $HOST.check.tmp
   % echo "Build area is available from host $HOST" > $HOST.check.tmp
   % rsh $HOST cat $BUILD/$HOST.check.tmp
   ```

   where $BUILD is the full path to the build area.

9. That $HOME is available from each host:

   ```
   % cd $HOME
   % rm $HOST.check.tmp
   % echo "HOME is available from host $HOST" > $HOST.check.tmp
   % rsh $HOST cat $HOME/$HOST.check.tmp
   ```

dmake Limitations

You can use any machine as a build server as long as it meets the following requirements:

- From the dmake host (the machine you are using to start the build process) you must be able to use rsh without being prompted for the password to remotely execute commands on the build server.
- The bin directory in which the dmake software is installed must be accessible from the build server. By default, dmake assumes that the logical path to the dmake executables on the build server is the same as on the dmake host. You can override this assumption by specifying a path name as an attribute of the host entry in the runtime configuration file.
- The /etc/opt/SPROdmake/dmake.conf file exists on the host, is readable, and contains the correct information. If this file does not exist, dmake will distribute only one job on this system

Installation

Running the non-GUI installer with the –extract-installation-data option can fail with no user-readable error message

Index