What's New in the Oracle® Solaris Studio 12.3 Release

Part No: E21987
January, 2012

ORACLE

120118@25097

Contents

Preface

This guide describes new and changed features, known problems, and limitations in Oracle Solaris Studio 12.3.

Supported Platforms

This Oracle Solaris Studio release supports platforms that use the SPARC family of processor architectures running the Oracle Solaris operating system, as well as platforms that use the x86 family of processor architectures running Oracle Solaris or specific Linux systems.

This document uses the following terms to cite differences between x86 platforms:

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

Information specific to Linux systems refers only to supported Linux x86 platforms, while information specific to Oracle Solaris systems refers only to supported Oracle Solaris platforms on SPARC and x86 systems.

For a complete list of supported hardware platforms and operating system releases, see the Oracle Solaris Studio 12.3 Release Notes.

Oracle Solaris Studio Documentation

You can find complete documentation for Oracle Solaris Studio software as follows:

- Product documentation is located at the Oracle Solaris Studio documentation web site, including release notes, reference manuals, user guides, and tutorials.
- Online help for the Code Analyzer, the Performance Analyzer, the Thread Analyzer, dbxtool, DLight, and the IDE 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.
- Man pages for command-line tools describe a tool's command options.

Resources for Developers

Visit the Oracle Technical Network web site to find these resources for developers using Oracle Solaris Studio:

- Articles on programming techniques and best practices
- Links to complete documentation for recent releases of the software
- Information on support levels
- User discussion forums.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

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%`
C shell for superuser	`machine_name#`

◆ ◆ ◆ CHAPTER 1

Introducing the Oracle Solaris Studio 12.3 Release

This 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.

What Is Oracle Solaris Studio?

Oracle Solaris Studio comprises a suite of tools for application development on Oracle 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 3.1 API for shared memory parallelization.
- The scriptable and multithread aware interactive dbx command-line debugger and dbxtool debugger GUI.
- The highly optimized and multithreaded Sun Performance Library.
- A Performance Analyzer to profile single- and multithreaded applications to detect performance bottlenecks and inefficiencies, and DLight for system profiling using DTrace technology on Oracle 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.
- New Code Analyzer tools for analyzing static code errors, dynamic memory access errors, and code coverage data together to find important errors in your code that cannot be found by other error detection tools.
- 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, performance 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 12.3 tools.

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

◆ ◆ ◆ CHAPTER 2

Compilers

This chapter describes the new and changed features in this release of the Oracle Solaris Studio C, C++, and Fortran compilers.

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.

- Support for new SPARC T4 platform: `–xtarget=T4`, `–xchip=T4`, `–xarch=sparc4`
- Support for new x86 Sandy Bridge AVX platform: `–xtarget=sandybridge –xchip=sandybridge –xarch=avx`
- Support for new x86 Westmere AES platform: `–xtarget=westmere –xchip=westmere –xarch=aes`
- New compiler option: `–g3` adds expanded debugging symbol table information.
- New compiler option: `–Xlinker` *arg* passes arg to linker, `ld`(1).
- The OpenMP default number of threads, `OMP_NUM_THREADS` is now 2 (was 1).
- Support for the 3.1 OpenMP shared memory parallelization specifications.
- Use `–library=sunperf` to link to the Sun Performance Library. This obsoletes `-xlic_lib=sunperf`.
- Support for user-supplied compiler option defaults.
- `–xarch` support for legacy SPARC architectures V7, V8, and V8a has been removed.

C Compiler

The following lists the new and changed features in this release of version 5.12 specific to the C compiler. For details, see the *Oracle Solaris Studio 12.3: C User's Guide* and the `cc` man page.

- A new suboption –`xbuiltin=%default` only inlines functions that do not set `errno`. The value of `errno` is always correct at any optimization level, and can be checked reliably.
- –`xkeepframe` option prohibits stack related optimizations for the named functions.
- Use of `-features=%none` and `-features=%all` deprecated in this release.
- New attributes `vector_size` and `returns_twice` recognized.
- `-xcheck=init_local` now initializes VLAs (variable length arrays) according to their base type.
- The functionality of the `aligned` attribute expanded to include automatics as well as globals and statics.
- `-xdumpmacros` provides information such as macro defines, undefines, and instances of usage.
- New option –`xanalyze={code|no}` produces a static analysis of the source code that can be viewed using the Oracle Solaris Code Analyzer.

C++ Compiler

The following lists the new and changed features in this release of version 5.12 specific to the C++ compiler. For details, see the *Oracle Solaris Studio 12.3: C++ User's Guide* and the `CC` man page.

- The g++ compatibility option, –`compat=g`, is available on Oracle Solaris x86 as well as Linux platforms.
- New compiler option: –`xivdep` sets the interpretation of `ivdep` pragmas. The `ivdep` pragmas tell a compiler to ignore some or all loop-carried dependences on array references that it finds in a loop for purposes of optimization. This enables a compiler to perform various loop optimizations such as microvectorization, distribution, software pipelining, etc., which would not be otherwise possible. It is used in cases where the user knows either that the dependences do not matter or that they never occur in practice.
- The –`compat=4` suboption ("compatability mode") has been removed. The default is now –`compat=5`. Also, `-compat=g` option for g++ source and binary compatibility, previously available only on Linux platforms, has been extended to Solaris/x86 as well.
- New option –`features=cplusplus_redef` allows the normally pre-defined macro `__cplusplus` to be redefined by a –`D` option on the command line. Attempting to redefine `__cplusplus` via a `#define` directive in source code is still not allowed. Also, use of –`features=%none` and –`features=%all` is now deprecated in this release.

- A new suboption –xbuiltin=%default only inlines functions that do not set errno. The value of errno is always correct at any optimization level, and can be checked reliably.
- C99 header stdbool.h and the C++ equivalent cstdbool are now available. In C++ the headers have no effect and are provided for compatibility with C99.
- New option –xanalyze={code|no} produces a static analysis of the source code that can be viewed using the Oracle Solaris Code Analyzer.

Fortran Compiler

The following lists the new and changed features in this release of version 8.6 specific to the Fortran compiler. For details, see the *Oracle Solaris Studio 12.3: Fortran User's Guide* and the f95 man page.

- Intrinsic routines LEADZ, POPCNT, and POPPAR previously had the return type as the type of the argument. In this release, to be conforming with the Fortran 2008 standard, the intrinsics return a default integer, regardless of argument type. This introduces a minor incompatibility with previous releases.
- Object Oriented Fortran features related to polymorphism are now supported:
  - Supported OOF features: type extension and polymorphic entities: CLASS statement, unlimited polymorphism, SELECT TYPE construct, ABSTRACT derived type, EXTENDS_TYPE_OF and SAME_TYPE_AS intrinsics, and sequence type assignments to unlimited pointers.
  - Unsupported OOF featutures: typebound procedures: typebound PROCEDURE declaration, GENERIC, DEFERRED, NON_OVERRIDABLE, PASS, NOPASS.
- Other new F2003/2008 features:
  - Enhanced structure constructor: using component names to construct a structure constant.
  - Enhanced PUBLIC/PRIVATE access control on module derived types and components.
  - More Fortran 2008 math intrinsic function support. Most Fortran 2008 math intrinsic functions are now supported except for ERFC_SCALED, NORM2 and some REAL*16 variants on x86 platforms.
  - Derived types with no components.
  - The KIND argument was added to ICHAR, IACHAR, ACHAR, SHAPE, UBOUND, LBOUND, SIZE, MINLOC, MAXLOC, COUNT, LEN, LEN_TRIM, INDEX, SCAN and VERIFY intrinsics.
  - The BACK argument was added to MINLOC and MAXLOC intrinsics.
  - New intrinsics FINDLOC and STORAGE_SIZE were added.
  - New keywords ERRMSG, SOURCE and MOLD were added to ALLOCATE statement, and ERRMSG was added to DEALLOCATE statement.

OpenMP

The following lists the new and changed features for the OpenMP shared memory API implemented by the C, C++, and Fortran compilers in this release. For details, see the *Oracle Solaris Studio 12.3: OpenMP API User's Guide*.

- The 3.1 OpenMP API specifications are fully supported in this release.
- The default for the `PARALLEL` and `OMP_NUM_THREADS` environment variables has changed to 2. It was 1 in previous releases. This means that autoparallelization and explicit OpenMP parallelization will get the benefit of two threads during execution by default. Previously, the default would have run these programs serially in one thread.
- Two new modes for binding threads to processors are available with the `SUNW_MP_PROCBIND` environment variable: `COMPACT` and `SCATTER`. See the *Oracle Solaris Studio 12.3: OpenMP API 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

Math library `libcx` has been removed in this release.

Sun Performance Library

This release of Sun Performance Library is available on the Oracle Solaris operating system and 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 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.
- 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 Sun Performance Library User's Guide* describes and shows examples for using the Sun Performance Library routines, the Fortran and C interfaces, optimization and parallelization options, the SPSOLVE and SuperLU sparse solver packages, and 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

- Improved BLAS performance on new Intel and SPARC platforms.

◆ ◆ ◆ CHAPTER 4

Code Analysis Tools

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

Discover

The following features were added in the Discover memory analysis tool in this release:

- The new `-a` option writes error data to the *binary_name.analyze*`/dynamic` directory for use by the Code Analyzer.
- The new `-F` option determines what happens if a binary you have instrumented with Discover forks while you are running it. By default, Discover continues to collect memory access error data from the parent process. If you want Discover to follow the fork and collect memory access data from the child process, specify `-F child` when you use the `discover` command to instrument your binary.
- The new `-b` option starts the specified browser automatically while running the instrumented binary.
- The new `-c` option lets you tell Discover to check for errors in all libraries, a specified library, or a list of libraries in a specified file.
- The new `-n` option tells Discover to not check for errors in executables.

For more information, see the `discover`(1) man page and the *Oracle Solaris Studio 12.3: Discover and Uncover User's Guide*.

Uncover

The following features were added to the Uncover code coverage tool in this release:

- The new `-a` option writes error data to the *binary_name.analyze*`/coverage` directory for use by the Code Analyzer.
- The new `-c` option turns on reporting of execution counts for instructions, blocks, and functions.
- The new `-o` option writes the instrumented binary file to the specified file.

For more information, see the `uncover`(1) man page and the *Oracle Solaris Studio 12.3: Discover and Uncover User's Guide*.

Code Analyzer

The new Code Analyzer tool lets you combine three types of analysis to help you produce secure, robust, and quality C and C++ applications. The Code Analyzer displays three types of data:

- Static code issues collected when you build your binary with the `-xanalyze=code` option
- Dynamic memory access issues detected when you instrument and run your binary with Discover, the Oracle Solaris Studio memory error discovery tool
- Code coverage issues detected when you instrument and run your binary with Uncover, the Oracle Solaris Studio code coverage tool

The Code Analyzer displays the analysis results, including a code snippet from the source file where each issue was detected with the relevant source line highlighted, an error path for a static issue, and Call Stack (and Allocated At Stack and Free Stack, if available) for a dynamic issue.

You can jump from a function call in the error path or stack to the associated source code line, find all usages of the function in your program, jump to the declaration of the function, and display a call graph for the function.

The Code Analyzer pinpoints the core issues in your code, those issues that, when fixed, are likely to eliminate the other issues.

For more information, see the online help in the Code Analyzer GUI, the *Oracle Solaris Studio 12.3 Code Analyzer User's Guide*,Oracle Solaris Studio 12.3 Code Analyzer Tutorial, and the `code-analyzer`(1) man page.

◆ ◆ ◆ CHAPTER 5

Performance Analysis Tools

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

Performance Analyzer

This section describes the new and changed features in this release of the Oracle Solaris Studio Performance Analyzer and related tools. For details, see the *Oracle Solaris Studio 12.3: Performance Analyzer* manual and the Help in the Performance Analyzer.

Changes to Performance Analyzer Tool

The Performance Analyzer tool features the following enhancements.

- Performance for processing large experiments, especially for Java experiments, is significantly improved.
- Many improvements were made for data filtering, described in "Filtering Enhancements" on page 23.
- Most data tabs now include context menus, which you open by right-clicking in the tabs. These context menus can be used to discover advanced tab-specific features, such as filtering.
- Experiment Comparison mode now supports comparing experiments on different executables and load objects. Source and disassembly for comparison is now shown in split panes for the two versions. Experiment Comparison mode can be enabled from context menus in the Functions and Source tabs.
- Quick navigation improvements in tabs enable you to do the following:
  - Double click on a function in the Functions tab to open the Source tab for that function.
  - Double click in a line in the Lines tab to open the Source tab at or near that line.

  - Double click on a line in the Source tab to open the Disassembly tab at or near the first instruction from that line.
  - Double click on a PC in the PCs tab to open the Disassembly tab at or near that address.
- The method used to search for source files referenced in experiments has changed. The pathmap is tried first, and then the search path combined with the pathmap is tried, then the original full path.
- Many improvements were made to the timeline, described in "Timeline Enhancements" on page 22.
- The Call Tree tab converts HW cycles to User CPU Time, and you can set the metric displayed in the Call Tree from the context menu.
- The Threads tab has an new display mode called Chart. When Chart is enabled for an experiment containing clock profiling data, the default Load Imbalance chart shows the total amount of CPU Time attributed to each thread.

Timeline Enhancements

The Timeline tab of the Performance Analyzer features the following enhancements:

- You can right-click on the Timeline to open a context menu to filter data, select events, zoom in and out, revert to the previous view, or change timeline properties.
- An event frequency chart, a line chart that displays frequency of events as a function of time. The chart is not displayed by default, and must be selected in the Set Data Preferences dialog box.
- An event state chart, a bar chart that shows the distribution of application time spent in various states as a function of time. For clock profiling data recorded on Oracle Solaris, the event state chart shows Oracle Solaris microstates. The chart is not displayed by default, and must be selected in the Set Data Preferences dialog box.
- Double-clicking in the Timeline now zooms in rather than opening the function color chooser dialog.
- A new Timeline Details tab replaces the Event tab. The new tab provides the event information as before, and also includes buttons for navigating the timeline, zooming, and changing function colors.
- Timeline now uses the font you specify when starting the Performance Analyzer.
- The timeline can be set to display aggregated events from each experiment. This is not a default setting and must be set be selecting Group Data by: Experiment in the Timeline tab of the Set Data Preferences dialog box.

Filtering Enhancements

Data filtering in the Performance Analyzer has been simplified and enhanced:

- Most data tabs now include context filters, which you select by right-clicking in the tabs. The names of new and existing context filters are also easier to understand.
- Context filters now filter data immediately and no longer require you to apply them in a separate dialog box. Selection of a context menu filter affects the data used by all tabs. Each filter selected is AND'ed with any existing filters, enabling you to progressively filter the data.
- The Filter Data dialog box was simplified and renamed to Manage Data Filters. However, filtering from the context menus in data tabs is the recommended way to filter data.

  The Custom tab of the Manage Data Filters dialog box can be used to view the current state of filters that were applied from context menus. The Custom tab allows user editing of the current filter expression. It also supports undo, redo, and Show Keywords, a dialog box that describes symbols that may appear in filtering expressions that are generated by the context menu filters.
- You can filter experiments using labels that you add with the new `er_label` command.
- The online help includes expanded information about filtering.

See "Filtering Data" in *Oracle Solaris Studio 12.3: Performance Analyzer* for more information.

New `er_label` command

The `er_label` command enables you to define part of an experiment and assign a name or label to it. The label captures the profiling events that occur during one or more periods of time in the experiment that you define with start time and stop time markers.

You can assign labels to experiments by running the `er_label` command at the command line or by executing it in scripts. Once you have added labels to an experiment, you can use the labels for filtering. For example, you might filter the experiment to include or exclude the profiling events in the time periods defined by the label.

One use of `er_label` is to support profiling a server program that is being driven by a client as an independent process or processes. In this usage model, you start the server with the `collect` command to start creating an experiment on the server. Once the server is started and ready to accept requests from a client, you can run a client script that makes requests to drive the server and runs `er_label` to label the portions of the experiment where the client requests occur.

See "Labeling Experiments" in *Oracle Solaris Studio 12.3: Performance Analyzer* for more information.

Changes to Experiments

The experiment format has changed and the version number is now 12.3, which matches the Oracle Solaris Studio version number.

The tools in Oracle Solaris Studio 12.3 can open experiments with the following version numbers:

- version 10.1, experiments created with Oracle Solaris Studio 12.2 or Sun Studio 12 update 1.
- version 10.2, experiments created with an early release of Oracle Solaris Studio 12.3 such as the Beta release.
- version 12.3, experiments created with the released version of Oracle Solaris Studio 12.3.

If you try to open experiments created from a prior release, you get an error saying that the experiment must be read with an earlier version of the tools.

Changes to Data Collection

Data collection changes affect the `collect` command, `dbx collector` command, and `er_kernel` command.

`collect` Utility

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

- `collect` does not verify that the target program is an ELF executable, which enables you to profile scripts without setting an environment variable. If the target is an ELF executable, `collect` checks it for compatibility with the machine on which it is running.
- Hardware counter support for SPARC T4, and Intel's Westmere and Sandy Bridge chips has been added for Oracle Solaris. Westmere support has been added for Linux, but Sandy Bridge support for Linux is not yet implemented.
- Hardware counter profiling now enables you to perform memoryspace profiling of any binary by prepending "+" to any precise hardware counter. Currently only SPARC T4 and T3 processors support precise hardware counters.

  To determine which hardware counters are precise, look for the keyword `precise` in the output of the `collect -h` command. To analyze memory access patterns in an experiment which includes memoryspace profiling data, select Memory Objects tabs such as Vaddress or Vline_64b in the Tabs tab of the Set Data Presentation dialog box. You can then use the context filters from the selected Memory Object tab to filter by data addresses.
- Hardware counter support on versions of Linux running the Linux kernel with a version greater than 2.6.32 is implemented with the `PerfEvents` framework, and does not require a kernel patch; for earlier systems, the `perfctr` patch is still required.

- Running `collect` without arguments now displays a usage message only. To display information about available hardware counters, you must run `collect –h` with no additional arguments.
- High-resolution clock profiling, which is performed using `collect -p high`, is now available on Linux systems that support it, including Oracle Linux 6.

dbx collector

The dbx collector is changed in this release as follows:

- Hardware counter support for SPARC T4, Westmere and Sandy Bridge chips has been added for Oracle Solaris. Westmere support has been added for Linux, but Sandy Bridge support for Linux is not yet implemented.
- Hardware counter support on versions of Linux running the Linux kernel with a version greater than 2.6.32 is implemented with the `PerfEvents` framework, and does not require a kernel patch; for earlier systems, the `perfctr` patch is still required.

`er_kernel` Utility

The `er_kernel` utility for profiling an Oracle Solaris kernel is changed as follows:

- `er_kernel` now enables you to perform profiling of the kernel and applications. You can use the `-F` option to control whether or not application processes should be followed and have their data recorded as sub-experiments to the kernel experiment.
- `er_kernel` no longer supports the `-T` option for profiling a specific process. You can use the `-F` option with a regular expression instead.
- The `er_kernel` utility can collect hardware counter overflow profiles for the kernel using the DTrace `cpc` provider, which is available only on systems running Oracle Solaris 11. You can perform hardware counter overflow profiling of the kernel with the `-h` option for the `er_kernel` command as you do with the `collect` command. However, dataspace profiling is not supported so dataspace requests are ignored by `er_kernel`.
- Running `er_kernel` without arguments now displays a usage message only. To display information about available hardware counters, you must run `er_kernel –h` with no additional arguments.
- If the hardware counter overflow mechanism on the chip allows the kernel to tell which counter overflowed, you can profile as many counters as the chip provides; otherwise, you can only specify one counter. The `er_kernel –h` output specifies whether you can use more than one counter by displaying a message such as "specify HW counter profiling for up to 4 HW counters."
- `er_kernel` does not verify that the target load is an ELF executable, which enables you to profile any command or script.

`er_print` Command

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

- Performance for processing large experiments, especially for Java experiments, is significantly improved.
- The method used to search for source files referenced in experiments has changed. The pathmap is tried first, and then the search path combined with the pathmap is tried, and then the original full path is tried.
- The `tlmode` subcommand can be used to set the default for the timeline to display aggregated events from each experiment.

Thread Analyzer

The following features were added or changed in Oracle Solaris Studio 12.3 Thread Analyzer.

- The Thread Analyzer now informs you if your application is not instrumented for race detection profiling
- The Thread Analyzer now opens the first call stack in the Races tab automatically.

DLight

DLight is an interactive graphical observability tool that uses Oracle Solaris Dynamic Tracing (DTrace) technology. DLight runs multiple DTrace scripts in a synchronized fashion, while showing you the output graphically, to help you trace a runtime problem in an application to the root cause.

The following features were added or changed in Oracle Solaris Studio 12.3 DLight.

- A new Process Tree target enables you to profile a process and all the processes that it creates, graphically showing the following:
  - An aggregation of the microstates of all the threads of all the processes that DLight profiled with the Process Tree Target.
  - The microstate transitions in the form of a timeline for each thread of the target process and its child processes.
  - Locking statistics for the process and its children.
  - Aggregated CPU usage of all threads in all the targeted processes profiled across all the CPUs they are running on.
  - CPU-intensive areas in your program's process tree by displaying the functions in your program along with the CPU time used by the function and any functions it calls.
- The Attach target, which is used to profile a single running process, was renamed to Process target.

- The AMP target was removed.

See the Help in DLight and the Oracle Solaris Studio 12.3: DLight Tutorial for more information.

◆ ◆ ◆ CHAPTER 6

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 12.3 dbx.

- `dbx` now includes macro expansion. For details, see Appendix C, "Macros," in *Oracle Solaris Studio 12.3: Debugging a Program With dbx*, or type `help macros` at the (dbx) prompt when running `dbx`.
- Object-Oriented Fortran Support:
  - `dbx` now supports type extension and polymorphic pointers. This is consistent with C++ support.
  - The `output_dynamic_type` and `output_inherited_members` dbx environment variables now also work with Fortran.
  - You can now use the `-r`, `+r`, `-d`, and `+d` options with the `print` and `whatis` commands to get information about the inherited (parent) types as well as the dynamic types.
- `dbx` now supports the Fortran allocatable scalar type, in addition to the allocatable array type.

◆ ◆ ◆ CHAPTER 7

The Oracle Solaris Studio IDE

The Oracle Solaris Studio 12.3 IDE (Integrated 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 12.3: IDE Quick Start Tutorial.

New and Changed Features

The following features were added or changed in the Oracle Solaris Studio 12.3 IDE:

- Based on NetBeans IDE 7.0.1
- The new C/C++ Project From Binary File project type lets you create a project from an existing binary by specifying the binary file, the location of the source files from which it was built, which files you want included in the project, and whether you want dependencies included in the project.
- You can now work on a project that resides on a defined remote host on your local host, including browsing and editing files on a remote host file system.
- You can open a terminal window for a remote host.
- The new template specialization functionality simplifies navigation between generic template and template specializations. You can use this navigation by right-clicking on the annotation icon in the margin of the Source Editor, or by pressing Ctlr+Alt and right-clicking on a template class or template method.

- You can create a project for an Oracle Database application. In order to do so, the Oracle Solaris Studio installation you are using must include the optional Oracle Instant Client component. The IDE now includes ProC support.
- The Source Editor does static code error checking on your project as you type and displays an error icon in the left margin when an error is detected.
- You can run Memory Access Error checking on your project.
- The New Run Command project property lets you specify the command and arguments that should be supplied to the project's command line when you run the project. The run command can be a shell script, or for a library project, it can be a binary.
- You now have the option of using the `gdb` debugger for code compiled using a gcc tool collection. The `dbx` debugger is the default.
- The new Desktop Distribution feature lets you generate a zip file containing a distribution of the IDE and the Code Analyzer that will run on almost any operating system and use the Oracle Solaris Studio compilers and tools on a remote server. When you run the IDE on the desktop system, it will recognize the server on which you generated the distribution as a remote host, and access the tool collection in your Oracle Solaris Studio installation.

For details, see the online help in the IDE and the Oracle Solaris Studio 12.3: IDE Quick Start Tutorial.

Software Requirements

The Oracle Solaris Studio IDE requires the Java SE Development Kit (JDK) 6 Update 24 or later. If the IDE cannot find the required JDK, it does not start and issues an error message.

Updating the IDE

Any updates to the Oracle Solaris Studio 12.3 IDE, dbxtool, DLight observability tool, and Code Analyzer will be delivered in Oracle Solaris Studio product patches, not through the NetBeans autoupdate feature, which is disabled by default in the IDE.

In the following cases, conflicts might occur in these tools when you install such product patches:

- If you have enabled the autoupdate feature in the tool and automatic updates have occurred.
- If you have installed plugins from the NetBeans Update Center.

To resolve the conflicts:

- If you installed your Oracle Solaris Studio tools with the package installer on Solaris 10 or from the IPS repository on Solaris 11, delete `ide-12.3-`*OS*`-`*architecture* (for the IDE or DLight), `dbxtool-12.3`–*OS*`-`*architecture*, or `code-analyzer–12.3`–*OS*`-`*architecture* from your Oracle Solaris Studio user directory at `~/.solstudio`.

- If you installed your Oracle Solaris Studio tools using the download tarfile, reinstall the tarfile.

Configuration

The default heap size for NetBeans IDE 7.0.1 is determined automatically, with respect to the amount of memory available on the system. The Oracle Solaris Studio 12.3 IDE generally runs well with the 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*`/lib/netbeans/etc/netbeans.conf file`, add the `–J-Xmx` command line Java startup switch, and then restart the IDE.

  For example:

```
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 Oracle 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-Oracle 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 8

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 Oracle 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 *Oracle Solaris Studio 12.3: Distributed Make (dmake)* manual.

The following features was added to the dmake utility in this release.

dmake can now use ssh in addition to rsh to remotely execute commands on the build server. If you want to use ssh, you must specify the remote path to the ssh command in your .dmakerc file.

The path to the remote shell can be specified in the .dmakerc file.

For example:

```
host earth { jobs = 3 }
host mars  { jobs = 5 , rsh = "/bin/ssh" }
```

If `rsh` = is not specified, `dmake` will use `/bin/rsh` by default.

As with `rsh`, you must ensure that `ssh` can login to the remote host without requiring a password, and does not issue any warnings or errors.

Software Corrections In This Release

- Fixed bug: `dmake` writes un-escaped :'s to `.make.state`, breaks itself.
- Fixed bug: Man page update: new option `-m grid`(SGE support).
- Fixed bug: `dmake` man page misses "distributed" mode in the SYNOPSIS section.
- Fixed bug: dmake ignores command line option `-x SUN_MAKE_COMPAT_MODE=GNU`.

Oracle Solaris Studio Installer

The new and changed features in the Installer include:

- The non-GUI installer now lets you specify which components to install.
- The new `-generate-desktop-dir` installer option lets you generate a zip file containing a distribution of the IDE configured for a desktop system with almost any operating system. After installing the Oracle Solaris Studio software, you can unzip this file on a desktop system. When you run the IDE on the desktop system, it will recognize the server on which you generated the distribution as a remote host, and access the tool collection in your Oracle Solaris Studio installation.
- The new `-nfs-server` installer option runs the installer in NFS server mode, which does not check the server for the required OS patches and does not install symbolic links to the Oracle Solaris Studio software and man pages in the `/usr/bin` and `/usr/share/man` directories.
- The new `-ignore-architecture` installer option lets you install the Oracle Solaris Studio components for SPARC based platforms on an x86 based platform, or the components for x86 based platforms on a SPARC based platform.
- The new `-force-uninstall` uninstaller option lets you force removal of the Oracle Solaris Studio packages and installation directory when the NBI registry is corrupted.

◆ ◆ ◆ CHAPTER 9

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. Any late-breaking issues are listed in the Oracle Solaris Studio 12.3 Release Notes.

Compilers

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

Issues Common To The Compilers

Documentation Errata

Errors in the published compiler documentation are listed below.

- The `cc`(1), `CC`(1), and `f95`(1) man pages neglect to list the `–xarch=sse3a` flag, which adds the AMD instruction set, including 3dnow, to the SSE3 instruction set.
- The C and C++ documentation neglects to point out that the `–xMF` option can only be used with `–xMD` or `–xMMD`, but not with `–xM` or `–xM1`. When specified, it overrides the default `.d` file name used with those options.

C++

Apache Standard Library Issue on Solaris

The Apache `stdcxx` library installed in Solaris 10u10 and earlier and in the initial release of Solaris 11 has a syntax error in header `stdcxx4/loc/_moneypunct.h`. This error was not seen by earlier compilers, but is caught by the Oracle Solaris Studio 12.3 C++ compiler. There is no way to disable the error detection.

A fix for this bug is available in a patch for Solaris 10, and in the first Solaris 11 SRU. The fix will be included in Solaris 10u11 and Solaris 11u1 when they become available.

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
```

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.

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.

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;
}
```

Fortran

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

- Blank space before the end of a no advance print line does not affect output position (7087522).

  Having the X edit descriptor at the end of a format of an output statement does not affect the position of the next character in the output record. This causes a difference if the output statement has **`ADVANCE='NO'`** and there are more characters to be transferred to the same record by subsequent output statements.

  In many cases, this can be worked around by having a blank character string edit descriptor instead of the *n***X** edit descriptor. They are not exactly the same since the blank character string edit descriptor actually causes blank characters to go into the record whereas the *n***X** only skips over the next *n* characters, usually causing blanks to be in those skipped positions by default.

- Valid code rejected when a line consists of two continuation ampersands. (7035243).

  An empty continuation line with a single ampersand is forbidden by the Fortran standard. However, with two ampersands on the same line, an empty continuation line can still be created without falling under the standard restriction. The compiler does not handle that case and gives an error. The workaround is to delete that line which only affects the readability of the program without adding any semantics.

- BOZ constants sometimes get truncated (6944225).

  In some relatively more complex scenarios, such as an array construct, a BOZ constant might get truncated to the default integer size of 4 bytes even though the corresponding item it is supposed to be assigned to is an 8-byte integer entity. The workaround is to use constants of correct type and kind in array construct instead of BOZ constants.

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

Fortran 77 Libraries Removed

Here is a reminder that the Oracle Solaris Studio 12.2 release removed the obsolete FORTRAN 77 libraries. This 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.

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.

gethrtime(3F) on Linux Platforms

There is no reliable way to accurately obtain the clock rate on AMD processors when system power saving is enabled. As a result, using timing functions based on `gethrtime`(3F) (the Fortran compiler's Linux version of the Solaris `gethrtime`(3C) function) to get high resolution real time on Linux platforms will only be accurate on AMD systems with power saving disabled. A reboot of the system might be required to disable the power-saving features.

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 12.3 `dbx` has the following limitations:

- 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 compiled with the `-compat=4` option. This problem does not occur for the `-compat=5` option.

Note – The `-compat=4` option was removed in this release. See "C++ Compiler" on page 14 for details.

- 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 the `-stackvar` option. 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 might 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.
- The following features of `dbx` are not available on the Linux OS:
  - Fix and continue
  - Performance data collection

- Breakpoints on the following events:
  - `fault`
  - `lastrites`
  - `lwp_exit`
  - `sysin`
  - `sysout`
  - `sync`
  - `throw`
- Java debugging
- Debugging 32–bit programs, unless you start `dbx` with the `-x exec32` option.

- The following problems may occur when debugging programs on Linux platforms:
  - `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
      ```
  - 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`.

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.

- `dbx` does not support the following features for the GNU C and C++ compilers:
  - VL array
  - Exception handling
  - OpenMP
  - RTTI
  - Template definition
  - Default argument
  - `using` directive
  - `friend`
- Some issues exist for `dbx` on Oracle Linux 6:
  - Indirect reference symbols used in system libraries can sometimes cause `dbx` to set breakpoints at the reference rather than the actual function.
  - When debugging code compiled with gcc 4.4.4 compilers:
    - `dbx` might hang when printing a variable length array.
    - `dbx` is unable to see local variables when stopped at the very end of a function (the `}` line).
    - `dbx` does not see macros defined using the `-D` compiler option.

Performance Analyzer

This section describes known problems with the Performance Analyzer tool.

- Sometimes the Callers-Callees tab shows a truncated metric value.
- The `icache` stall time might be overestimated on SPARC T4 processors.
- The OpenMP Wait metric for OMP tasks and OMP parallel regions do not add up to the OpenMP Wait metric for `<Total>` because any profiling packets that arrive before the first parallel region entry do not get counted as OMP Wait time for any task or region, but do get counted in the total.
- Adding and dropping experiments is not always properly handled; the workaround is to load all experiments initially, and filter to exclude the ones you would want to drop.
- Moving tabs does not always work properly.
- Multiple selection of items in tabs does not always work properly.

- Analyzer sometimes hangs at SummaryDisp.updateSummary(SummaryDisp.java:249)
- The Summary tab is not updated when a selection is made in the Source-Disassembly tab.
- Highlighting in the margin of the Source tab may not be shown even when there are non-zero metrics.
- The Show/Hide/API-only functionality for shared objects does not always work properly in conjunction with filtering. This issue also occurs when using `er_print` to view experiment data.
- The Apply button in the General tab of the Manage Filters dialog box does not always apply changes. Pressing Apply more than once may help. A more reliable workaround is to access filters from the context menus in the Timeline, Threads, and CPUs data tabs.
- In the Timeline tab, Threads and CPUs might not be shown in numerical order.

collect Utility

This section describes known problems with the `collect` utility and data collection.

- For some optimized codes, the stack unwind on Sandy Bridge machines is sometimes incorrect.
- Because of an implementation change for locking algorithms in JDK 1.7, synchronization tracing on Java programs double-counts all Java synchronization events. The workaround is to divide the reported values by two.
- On Linux systems, clock-profiling of multithreaded applications will report inaccurate data for threads. The profile signal is not always delivered by the kernel to each thread at the specified interval; sometimes the signal is delivered to the wrong thread. If available, use hardware counter profiling using the `cycle` counter for more accurate per-thread results.
- On systems with multiple CPUs running at different clock frequencies, clock profiling will generate events based on the clock rate for one CPU, which can lead to over- or under-counting for events on other CPUs.

Thread Analyzer

This section describes known problems with the Thread Analyzer.

There might be a runtime failure of the Thread Analyzer, if ALL of the following is true:

- `collect` is run on a binary instrumented with `discover` for data race detection
- the run happens a machine with SPARC processor that supports hardware capability filters, such as the UltraSPARC T2
- the machine is running Solaris 10 Update 9 or an earlier update

To avoid the problem, before you run `collect`, set an environment variable `LD_NOAUXFLTR=yes` to skip loading the filter library. There might be some performance impact due to not using filters.

er_kernel Utility

This section describes known problems with the `er_kernel` utility:

- `er_kernel` sometimes can crash the machine on which it is run. The problem has only been seen on Oracle Solaris 10 running on UltraSPARC T1, T2, and T2+ chips and is due to a hypervisor bug.
- `er_kernel` profiles sometimes misattribute CPU state, and do not correctly report user data.
- Sometimes one or more CPUs will stop generating `er_kernel` events for 30 seconds or more.
- DTrace stack unwind for user processes sometimes omits one frame, typically when the leaf function is executing within its function prologue or epilogue.
- On systems with multiple CPUs running at different clock frequencies, `er_kernel` profiling generates events based on the clock rate for one CPU, which can lead to over-counting or under-counting for events on other CPUs.

IDE

This section describes known problems in the IDE.

- Versioning framework does not work in full remote mode.(195121)

  The versioning framework often works in terms of `java.io.File`, which makes it impossible to create a plugin capable of working with remote file objects.

  Workaround: Use versioning tools directly on the remote host through ssh.
- On some platforms, where GDB 7.2 is used, "Step Over" sometimes behaves as "Continue". (200196)

  Workaround: Try an earlier GDB version or change the Console Type in the Run section of the Project Properties from Internal Terminal to another option.
- Memory Access Error tool does not work for remote projects. (7109562)

  If you create a project on a remote host, and then instrument and run the project for memory analysis, you might receive the error message `can't execute: discover: No such file or directory`.
- Clicking the Debugger Console tab does not set focus to the debugger command prompt. (7102076)

  Workaround: Click a second time in the tab to set focus so that you can type a command at the prompt.

- Newly created full remote project from binary does not appear in the Projects tab. (7110094)

  When you are running a desktop distribution of the IDE and create a project from an existing binary on a remote host, the newly project does not appear in the Projects tab.

  Workaround: Choose File > Open Remote C/C++ Project and select the project to open.
- Can't create full remote project from binary on SPARC platform because binary is not recognized. (7109551)

  When you are running a desktop distribution of the IDE and create a project from an existing binary on a remote host that is a SPARC platform, and then choose File > Create Remote C/C++ Project, the binary is not recognized. If the filter in the file chooser is set to All Binary Files, the binary is not displayed; if the filter is set to All Files, you can select the binary, but will get the message "Binary file not found."

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` or `ssh`) 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` or `ssh` 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.
- In some cases, if you run the `register_solstudio` utility in your installation directory, it does not generate a registration page and open it in your browser.

  Workaround:

  1. Copy the `register_solstudio` utility from *installation_directory*`/bin` to *installation_directory*`/bin/condev/bin`.
  2. Replace *installation_directory*`/bin/register_solstudio` with a symbolic link to *installation_directory*`/bin/condev/bin/register_solstudio`.
  3. Run the `register_solstudio` utility and it will generate a registration page and open it in your browser.

Index