What's New in Oracle® Solaris Studio 12.4

Part No: E37071
December 2014

ORACLE®

Contents

Using This Documentation

- **Overview** – Describes the new and changed features in the compilers and tools with this Oracle Solaris Studio 12.4 release
- **Audience** – Application developers, system developers, architects, support engineers
- **Required knowledge** – Programming experience, software development testing, aptitude to build and compile software products

Product Documentation Library

For more information about compilers and tools, see the relevant man pages or Help and the documentation library at http://docs.oracle.com/cd/E37069_01.

System requirements and known problems are included in the "Oracle Solaris Studio 12.4: Release Notes ".

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.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

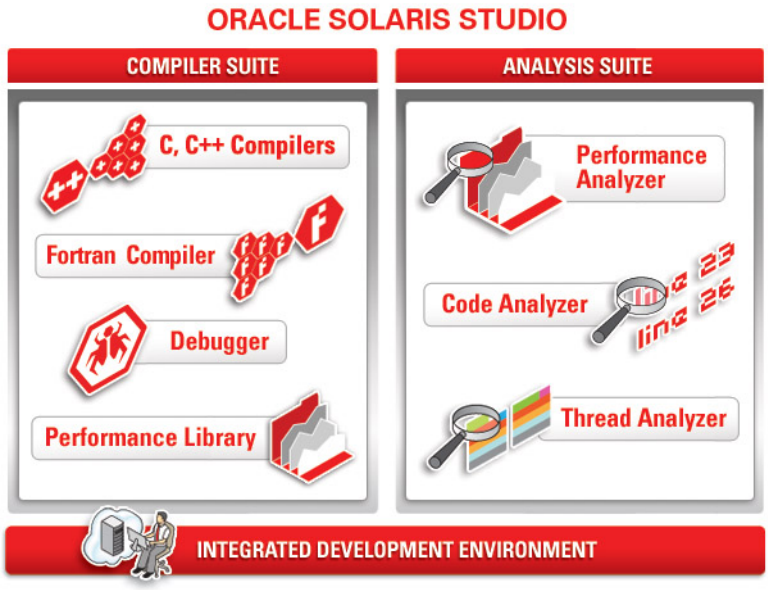
♦ ♦ ♦ CHAPTER 1

Introducing the Oracle Solaris Studio 12.4 Release

This chapter provides an overview of the key updates in this release.

Overview of Oracle Solaris Studio

Oracle Solaris Studio includes a compiler suite, an analysis suite, and a graphical integrated development environment (IDE) that is tailored for use with the compilers and tools from both suites. Together they provide a development environment that is optimized for developing applications with the best performance on Oracle Sun hardware.



Key Features in this Release

Oracle Solaris Studio 12.4 includes enhancements to all its compilers and tools. The key features are in the following areas:

C++ Compiler

Support for the C++11 Standard. See Chapter 2, “C++ Compiler”.

Performance Analyzer

Redesigned graphical interface to make it easier to navigate and understand performance data and specify which portions of your application to analyze.

Ability to run Performance Analyzer on a Linux, Windows, or Mac client and remotely access performance data on an Oracle Solaris *or Linux* system.

See Chapter 3, “Performance Analysis Tools”.

Code Analyzer

Significant improvements in eliminating false-positives during static analysis.

Reduced overhead time when collecting run-time data.

See Chapter 4, “Code Analysis Tools”.

Debugger

Faster `dbx` start-up time when processing large binaries.

	Reduction in the size of debugging information. See Chapter 5, “Debugging Tools”.
IDE	Significant memory footprint reduction in the IDE, so large projects can be opened, searched, and modified faster. Project settings that make large projects easier to use in version control systems. See Chapter 6, “Oracle Solaris Studio IDE”.
Optimization for new servers	Compiler and library optimizations for application performance improvements for Oracle Sun hardware servers: SPARC T5, SPARC M5, SPARC M6, SPARC M10, SPARC M10+, Intel Haswell, and Intel Ivy Bridge. Note that some of these improvements were already introduced in Oracle Solaris Studio 12.3 Platform-Specific Enhancements releases. See “Application Performance on New Hardware” on page 69.
OpenMP 4.0	Implementation of new features in the OpenMP API Version 4.0, a major upgrade to the OpenMP API standard language specification. See Chapter 7, “OpenMP API and Thread Analyzer”.

♦ ♦ ♦ CHAPTER 2

C++ Compiler

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

About the C++ Compiler

The Oracle Solaris Studio C++ compiler enables you to build high-performance parallel C++ applications for the newest SPARC processor-based systems from Oracle as well as Intel x86 processor-based systems.

The C++ compiler (`CC`) produces code that is targeted for specific operating systems, processors, architectures, memory models (32-bit and 64-bit), floating-point arithmetic, and more, according to command-line options you specify. The compiler automatically parallelizes serial source code to produce binaries with better performance on multicore systems and can also prepare binaries for enhanced debugging or analysis by other Oracle Solaris Studio tools. The compiler also supports GNU C/C++ compatibility features.

Support for the C++11 Standard

The new C++11 standard strengthens C++, giving you additional tools to help you make your code even cleaner and safer. The compiler retains accelerated SPARC and x86 performance on Oracle hardware. New features like auto, smart pointers, nullptr, range for, nonmember begin and end, lambda functions and algorithms, rvalue references, "move" constructors, uniform initialization and initializer lists are expected to change the way C++ libraries are designed.

This is the first Oracle Solaris Studio release to include support for the C++11 standard. All features of C++ 11 are supported in this release except as noted below:

No support for the following features:

- C++ 11 concurrency and atomic operations
- User-defined literals

Using C++11 Features

In Oracle Solaris Studio 12.4, the C++ compiler supports C++11, a new language and ABI (Application Binary Interface).

In C++ 11 mode, the CC compiler uses the g++ ABI and a version of the g++ runtime library that is supplied with Oracle Solaris Studio. For this release, version 4.8.2 of the g++ runtime library is used.

An ABI describes the low-level details in the generated object code. Modules that use different ABIs cannot successfully be linked together into a program. This means that you must use C++11 mode on all modules in your program, or none of them.

If you use Oracle Solaris Studio 12.4 C++ as an upgrade to Oracle Solaris Studio 12.3 (C++ 5.12), no changes are needed in scripts or makefiles if you are not using C++11 features. An exception is Rogue Wave Tools.h++ is not available. For more information about features no longer supported, see “Features That Have Been Removed in This Release” in “Oracle Solaris Studio 12.4: Release Notes ”

To compile in C++11 mode, add the option `-std=c++11` to the `CC` command line. The location on the command line is not important. The option causes the compiler to recognize language features new in C++11, and to use the C++11 version of the standard library (g++ runtime library that is supplied). Except for the options marked as incompatible with `-std=c++11`, all other command-line options can be used along with C++11, and have their usual effects. The `-std=c++11` option must be used consistently on every `CC` command used in building a library or executable program.

Note - No C++11 features are available by default. To use any C++11 features, you must use the new `-std=c++11` option with the `CC` compiler. This option uses g++ ABI, and there is no option to pick a different ABI. The option must be used to compile all modules of a program.

Compatibility Information for C++11 in this Release

The C++ compiler in this release has the following updated version details.

Compiler version:	C++ 5.13
Compiler version macro:	`__SUNPRO_CC = 0x5130`

The compiler version macro is strictly greater than all earlier releases, so version comparisons such as `__SUNPRO_CC>=0x5100` continue to work.

Compiler default mode: C++03 with `-compat=5`.

This is the same default mode as C++ 5.12 in the Oracle Solaris Studio 12.3 release.

Oracle Solaris Studio 12.3 had the following compiler mode options:

`-compat=5` This option selects C++03 with the Sun ABI. This is the default.

`-compat=g` This option selects C++03 with the g++ ABI, using the gcc headers and libraries provided with the compiler. In Oracle Solaris Studio 12.3, the gcc runtime libraries are installed in `/usr/sfw/lib` on Oracle Solaris if present. In this release, gcc runtime libraries supplied with the compiler are used instead.

The release adds the option `-std=[ c++11 | c++0x | c++03 | sun03]` where the option values are defined as follows:

`-std=c++11` This option selects C++11 with the g++ ABI and uses the g++ 4.8.2 runtime libraries installed as part of Oracle Solaris Studio 12.4.

`-std=c++0x` This option is equivalent to the `-std=c++11` option and is provided for GCC compatibility. The C++11 standard was nicknamed C++0x initially.

`-std=c++03` This option is equivalent to the `-compat=g` option.

`-std=sun03` This option is equivalent to `-compat=5`

Note - You cannot mix `-compat` and `-std` options and will get an error if you use both. If more than one `-std` option or more than one `-compat` option appears on a command line, the last one specified overrides the ones specified earlier. For example:

```
-compat=g   -compat=5   // OK, -compat=5 is used
-std=c++11  -std=c++03  // OK, -std=c++03 is used
-std=c++11  -compat=g   // always an error
-compat=g   -std=c++03  // always an error
```

Incompatibility of 16-bit Unicode with C++11

The option `-xustr=ascii_utf16_ushort` is not compatible with C++11, and is not allowed.

The option interprets `U"`*ASCII_string*`"` as 16-bit Unicode, but C++11 requires 32-bit Unicode for that syntax.

Library Incompatibilities with C++11 in this Release

With `-std=`*x* or `-compat=g`, the following `-library=`*v* options are not allowed:

- `Cstd`
- `stlport4`
- `stdcxx4`
- `Crun`
- `iostream`

When you need to list libraries to be linked, such as when you are creating a shared library, use the following options in this order when using the `-compat=g` or `-std=`*x* options:

```
-lstdc++ -lgcc_s -lCrunG3
```

When creating an executable program using `CC`, you should not list these libraries, because the `CC` driver will list them for you.

For information about libraries removed in this release, see "Features That Have Been Removed in This Release" in "Oracle Solaris Studio 12.4: Release Notes ".

Example Using C++11 Mode

The following commands could be used for building executable program `myprogram` from modules `main.cc`, `f1.cc`, and `f2.cc`:

```
% CC -std=c++11 -m32 -O -c main.cc
% CC -std=c++11 -m32 -O -c f1.cc f2.cc
% CC -std=c++11 -m32 -O main.o f1.o f2.o -o myprogram
```

If you have a Makefile that builds a C++ program using older versions of Oracle Solaris Studio, you can convert it to build a C++11 program by adding `-std=c++11` to every `CC` command line (typically in the `CCFLAGS` and `LFLAGS` macros), and by removing any incompatible options such as `-compat=5` or `-library=stlport4`. Valid C++ programs usually compile and run unchanged when compiled in C++11 mode. Unfortunately, many real programs depend, sometimes accidentally, on non-standard compiler behavior or extensions. Such code might not compile in C++11 mode.

Additional C++ Compiler Changes

The following lists the new and changed features in this release of version 5.13 specific to the C ++ compiler. For more information, see the `CC` (1) man page.

The C++ compiler changes include the changes that are described in "New and Changed Features Common to the Compilers" on page 69.

For details, see the "Oracle Solaris Studio 12.4: C++ User's Guide " and the `CC` man page.

- Support for `-compat=g` on all platforms .
- New compiler option: `-std` enables selection of either C++ 03 or C++ 11 dialect with g++ binary compatibility. When using `-std=c++11`, the following limitations apply:
  - Universal Character Names (escaped Unicode characters) are not currently supported.
  - The non-standard `iostream` headers ending in `.h`, such as `<iostream.h>`, `<fstream.h>`, etc., are not available. These headers were intended to ease the transition from old-style C++ to C++98.
- New compiler option: `-features=[no%]rtti` disables runtime type identification (RTTI).
- New compiler option: `-xprevise` produces a static analysis of the source code that can be viewed in Code Analyzer.
- The following options that have `-x`*option* equivalents are now deprecated:

  ```
  -help
  -inline
  -libmieee
  -libmil
  -nolib
  -nolibmil
  -pg
  -time
  -unroll
  ```

  Instead you should use `-xhelp`, `-xinline`, `-xlibmieee` and so on.
- Support for `-xregs=float` on x86.
- Behavior for `-errtags` is now the same as for the C compiler, emitting tags only for warning messages. In previous C++ compilers, the `-errtags` option caused a tag to be printed as part of the message for both warnings and errors.
- Default `-template` option changing from `-template=extdef` to `-template=no%extdef`.

  This change is because no other compiler uses the `definitions separate` template model that is assumed by `-template=extdef`. The `-template=extdef` option imposes strict requirements on how source code is organized, which most code does not follow. Unless you develop only with Oracle Solaris Studio C++, it is likely that the `-template=no%extdef` option is needed.

  For more information, see Chapter 6, "Creating and Using Templates," in "Oracle Solaris Studio 12.4: C++ User's Guide " and `Understanding the Effects of the Changed Default C++ Template Compilation Model (http://www.oracle.com/`

technetwork/articles/servers-storage-dev/changed-default-cpp-template-model-2292727.html).

Note - The `-library=stdcxx4` option does not currently work with `-template=no%extdef`. Specify `-template=extdef` when compiling C++ code on the command line if using the `-library=stdcxx4` option, until a patch for the library is available.

Stricter C++ Rules Enforced By Compiler

The C++ compiler enforces some C++ rules more strictly than past compilers. For explanations of the stricter enforcement, examples of offending code, and solutions to correct the offending code, see "New Features and Functionality of the Oracle Solaris Studio 12.4 C++ 5.13 Compiler" in "Oracle Solaris Studio 12.4: C++ User's Guide ".

♦ ♦ ♦ CHAPTER 3

Performance Analysis Tools

The performance analysis tools work together to enable you to analyze your application's behavior and find trouble spots that impact performance.

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

About Performance Analyzer

Performance Analyzer provides insight into the behavior of your application to enable you to find problem areas in your code. Performance Analyzer identifies which functions, code segments, and source lines are using the most system resources. Performance Analyzer can profile single-threaded, multithreaded, and multi-process applications, then present the profiling data to help you identify where you can improve your application's performance.

Performance Analyzer consists of a set of commands and tools including the `collect` utility which gathers profiling data, the `er_print` utility which displays interpreted profiling information in text form, and the Performance Analyzer GUI, which presents profiling information graphically.

Thread Analyzer is a related tool that enables you to focus on multithreading problems. See "Thread Analyzer" on page 67 for more information.

Documentation for Performance Analyzer

This release includes the new "Oracle Solaris Studio 12.4: Performance Analyzer Tutorials " manual which includes step-by-step instructions for profiling sample applications and examining data experiments in Performance Analyzer.

The "Oracle Solaris Studio 12.4: Performance Analyzer " reference manual contains details about Performance Analyzer, `collect`, `er_print` and other utilities.

Performance Analyzer New Features

This section summarizes the new features in this release of the Performance Analyzer and related tools. For more information, see the Help in Performance Analyzer and man pages for each command-line tool.

- The Performance Analyzer UI has been redesigned with new navigation features, new Overview and Welcome screens. See "User Interface Redesign" on page 20.
- The timeline features improved data display, navigation, and filtering. See "Timeline Improvements" on page 23.
- Source and Disassembly data views have syntax highlighting and hyperlinks for navigation. See "Source and Disassembly Improvements" on page 25 for more information.
- Call Tree view has new graphic indicators and navigation improvements. See "Call Tree Improvements" on page 26 for more information.
- Profiling of Java applications is easier to perform. See "Java Profiling Improvements" on page 27.
- Data collection and presentation for memoryspace profiling has been improved. See "Memoryspace Profiling Improvements" on page 28.
- New metrics can be calculated in hardware counter experiments. See "New Derived Metrics: CPI and IPC" on page 29 for more information.
- Experiments recorded on any platform can be read with the Performance Analyzer or `er_print` utility running on any other platform. See "Cross-Platform Analysis" on page 30.
- Performance Analyzer can now be used remotely from Windows or Mac machines, or from Oracle Solaris or Linux machines that do not have a full Oracle Solaris Studio installation. See "Remote Use of Performance Analyzer" on page 30.
- The tools now support I/O tracing of the target program. See "New I/O Data View" on page 32.
- A new Heap view provides improved display of heap tracing data to help find memory leaks. See "New Heap Data View" on page 33.
- "Other Changes to Performance Analyzer" on page 34

For details, see the Help in Performance Analyzer.

User Interface Redesign

Performance Analyzer user interface features improved data presentation and navigation. Some highlights of the UI changes include:

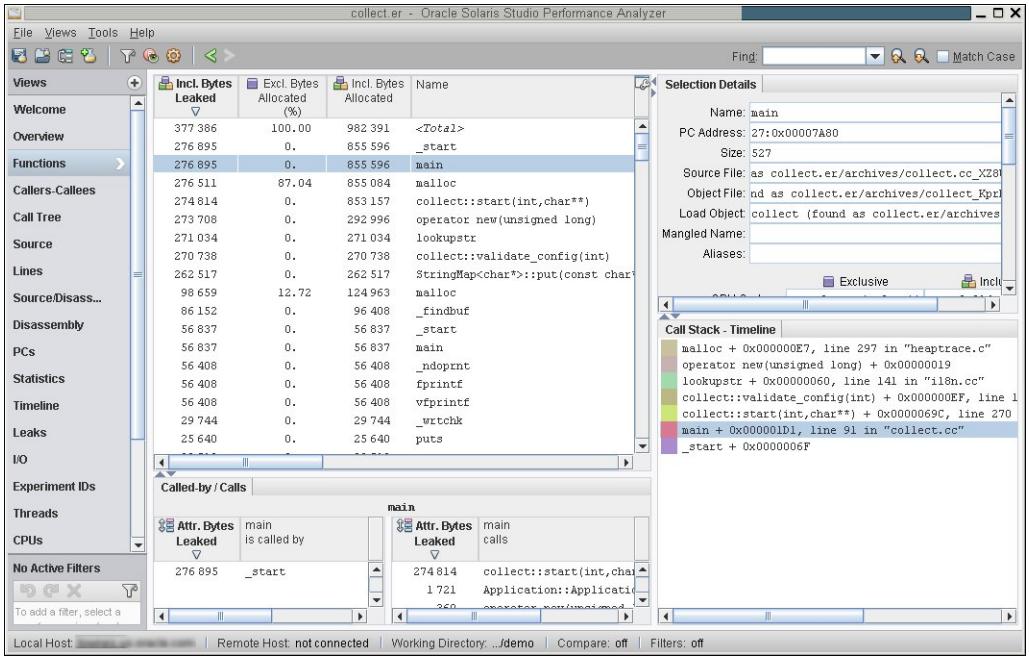
- Performance data is now shown in data views that you access through a navigation bar on the left. This replaces the horizontal tab navigation that was used previously. See "Performance Analyzer Navigation" on page 21 for more information.

- A new Welcome screen displays when you start Performance Analyzer without specifying an experiment name. See "Welcome Screen of Performance Analyzer" on page 22 for more information.
- A new Overview screen displays as the first view of the data when you open an experiment. See "Overview Screen of Performance Analyzer" on page 23 for more information.

Performance Analyzer Navigation

Performance Analyzer is now organized around data views that you access from a navigation bar on the left side of the Performance Analyzer window. Each view shows a different perspective of the performance metrics for your profiled application. As before, the Functions view serves as a focal point; when you select a function the other data views are updated to also focus on that selected function.

FIGURE 3-1 Vertical Navigation Area with Functions View Selected



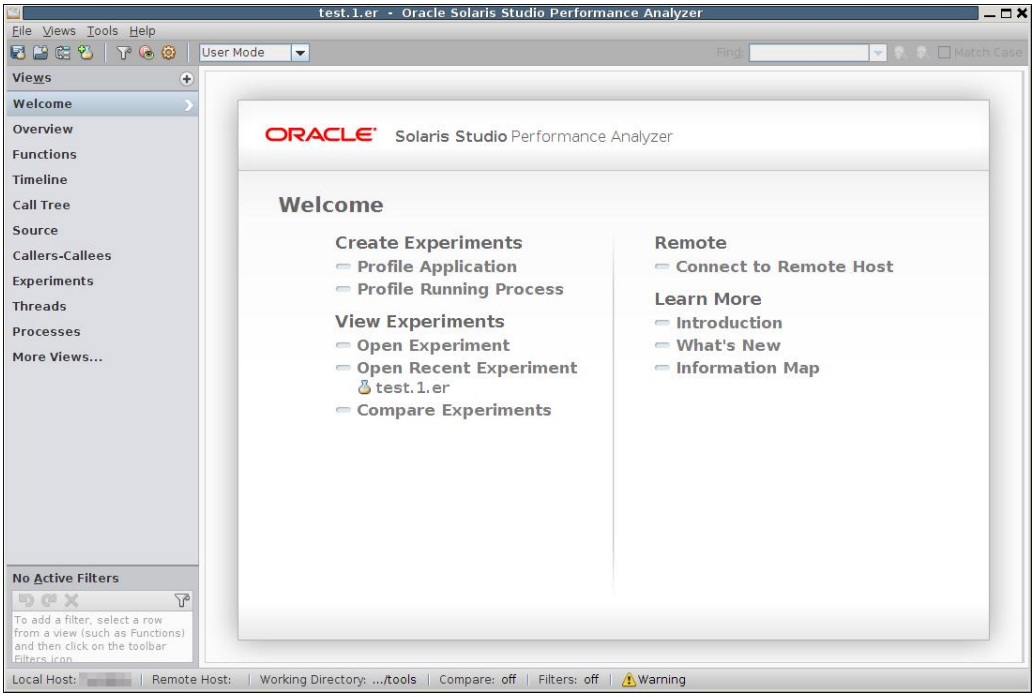
The data views available in the navigation bar are determined by the data available in the experiments you open. You can easily add and remove data views using the Views menu or the "+" button above the navigation bar. Details about items you select in the data view are shown on the right side, as you can see in the preceding figure.

Below the navigation bar, the Active Filters area shows the names of data filters that are currently applied and enables you to remove and reapply filters or remove all filters. You can apply filters by right-clicking an item in a data view and selecting a filter from the popup list or by clicking the filter icon in the toolbar.

Welcome Screen of Performance Analyzer

Performance Analyzer displays a new Welcome screen when you do not specify the name of an experiment when you start the tool. The Welcome screen provides links to make it easier to profile your application, view recent experiments, browse experiments, and view documentation.

FIGURE 3-2 Welcome Screen of Performance Analyzer



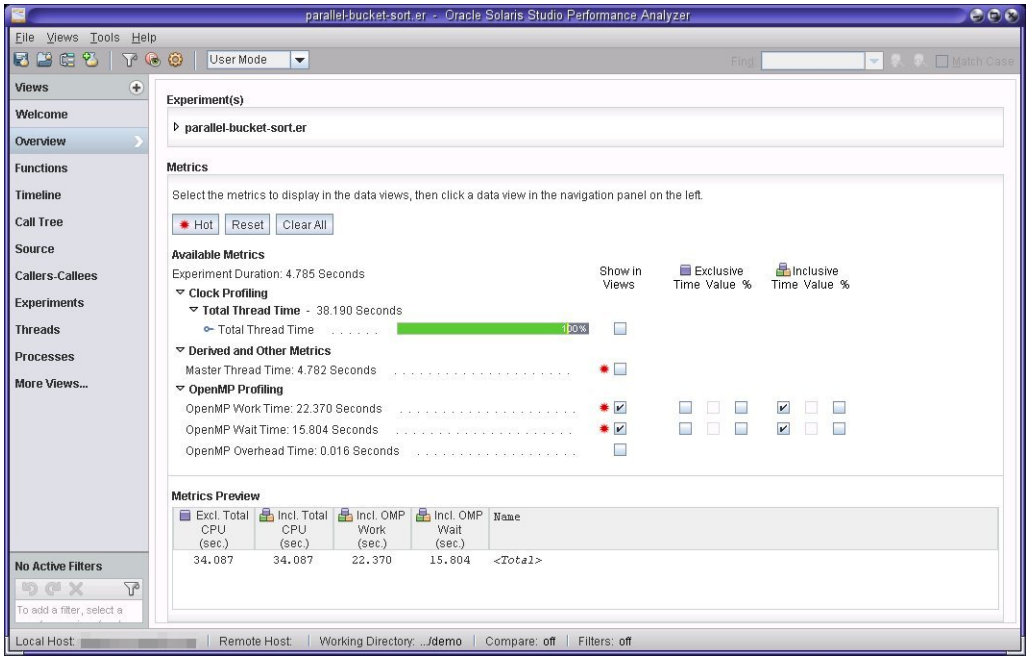
The Welcome screen is always available from the navigation bar so you can use it as a home page for using the Performance Analyzer, where you can click a link to perform an activity or open the Performance Analyzer Help viewer to read documentation.

Overview Screen of Performance Analyzer

The Overview shows performance metrics for the loaded experiments. Use the Overview to select the metrics you want to explore in other data views.

For each metric a value is displayed that represents the total for all loaded experiments. Colored bars are displayed to show relative values of related metrics. A highlight marker for high activity metrics indicates a metric that showed significant activity when compared to CPU time. These metrics can help identify problem areas in your program.

FIGURE 3-3 Performance Analyzer Overview and New Navigation



See the Help in the Performance Analyzer for more information about the Overview screen.

Timeline Improvements

The Timeline view has the following enhancements to make it easier to examine your data.

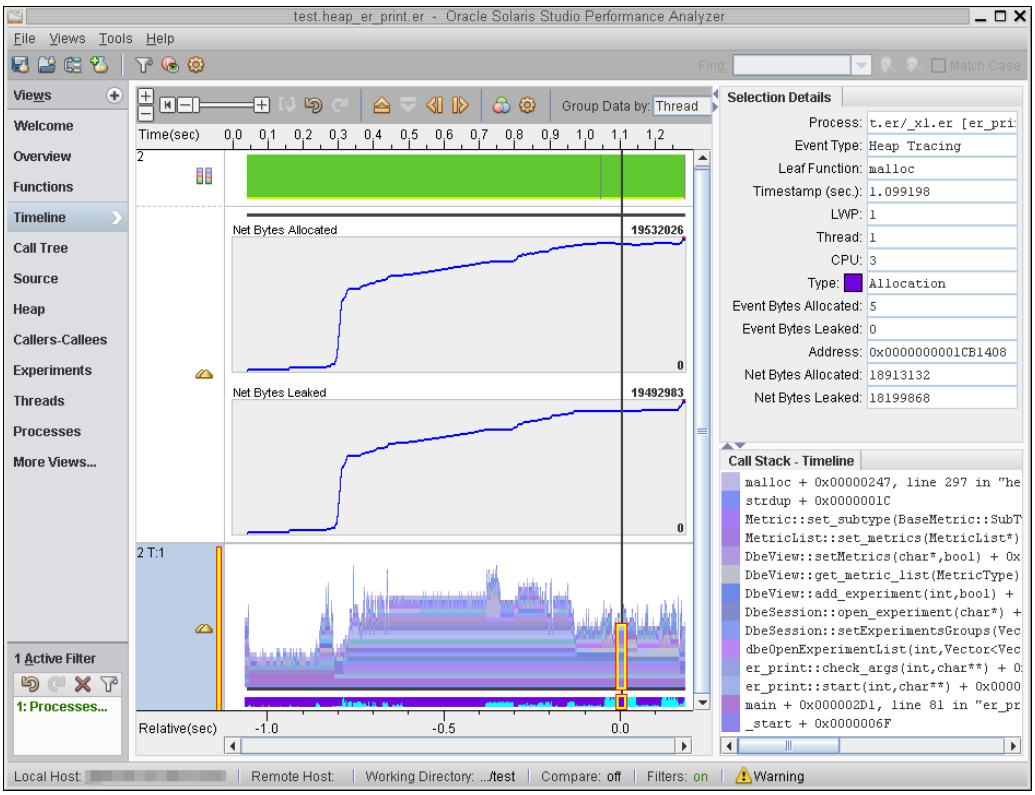
- Click and then drag the mouse to set markers that define a **precise** time period for zooming and filtering.

- Zoom to show the maximum number processes and threads using the new toolbar control for vertical zoom.
- Zoom in using Shift-drag with the mouse.

See the Timeline section of the **Keyboard Shortcuts and Mnemonics** in the Help menu for more information about navigation in the timeline.

The following figure shows a Timeline view of heap tracing data. The timeline shows graphs of heap allocations and leaks, as well as the callstacks for the allocations and frees. The right side shows details about the current selection's memory allocation and callstack.

FIGURE 3-4 Timeline View



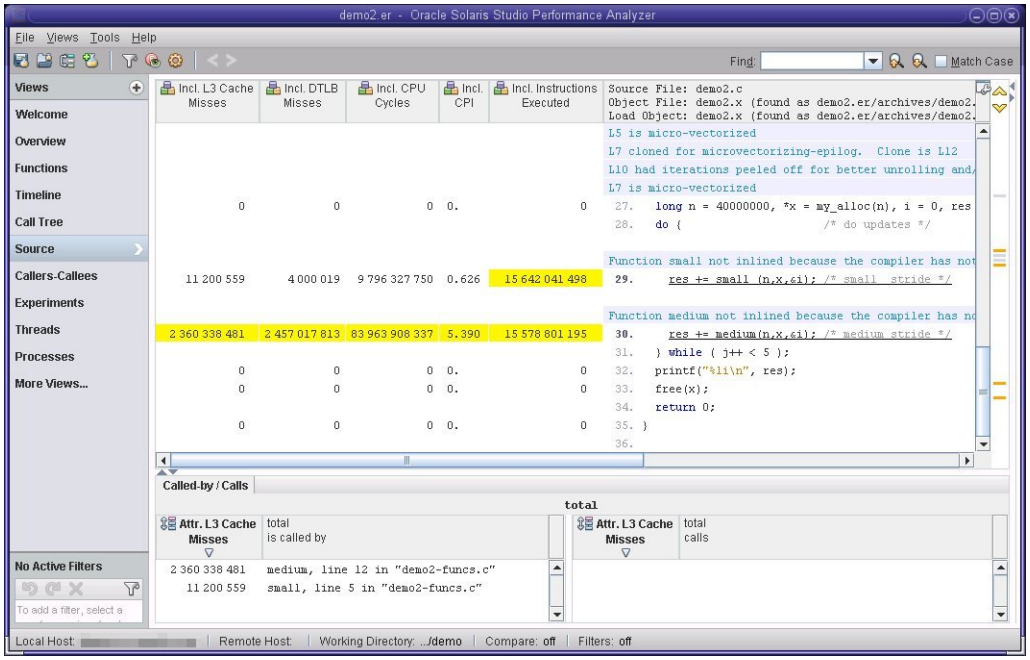
See "Oracle Solaris Studio 12.4: Performance Analyzer Tutorials " for step-by-step instructions for using the Timeline on sample experiments.

Source and Disassembly Improvements

The Source view is now shown with syntax highlighting based on the source language and includes many navigation improvements including hyperlinks for caller and callee functions.

The following figure shows the Source view displaying hardware counter metrics attributed to two lines of source code.

FIGURE 3-5 Syntax Highlighting and Navigation Improvements in Source Data View



Other changes to the Source and Disassembly views include the following:

- Navigation controls in the right margin help you jump to lines with high metrics.
- Right-click menus and hyperlinks let you navigate easily between Source and Disassembly views of the callers and callees of functions.
- Called-by/Calls tabs at the bottom of Source and Disassembly views enable you to navigate call paths. Select a function in the view and then use these tabs to navigate to functions it was called by, or to function calls it makes. When you click a function in Called-by/Calls tabs, the function is selected in the data view.

- Source view displays a warning if the source file is newer than the experiment.
- Forward and back buttons to enable you to navigate the history of actions you perform in the Source or Disassembly view.

See the Help in the Performance Analyzer for more information about the Source and Disassembly views.

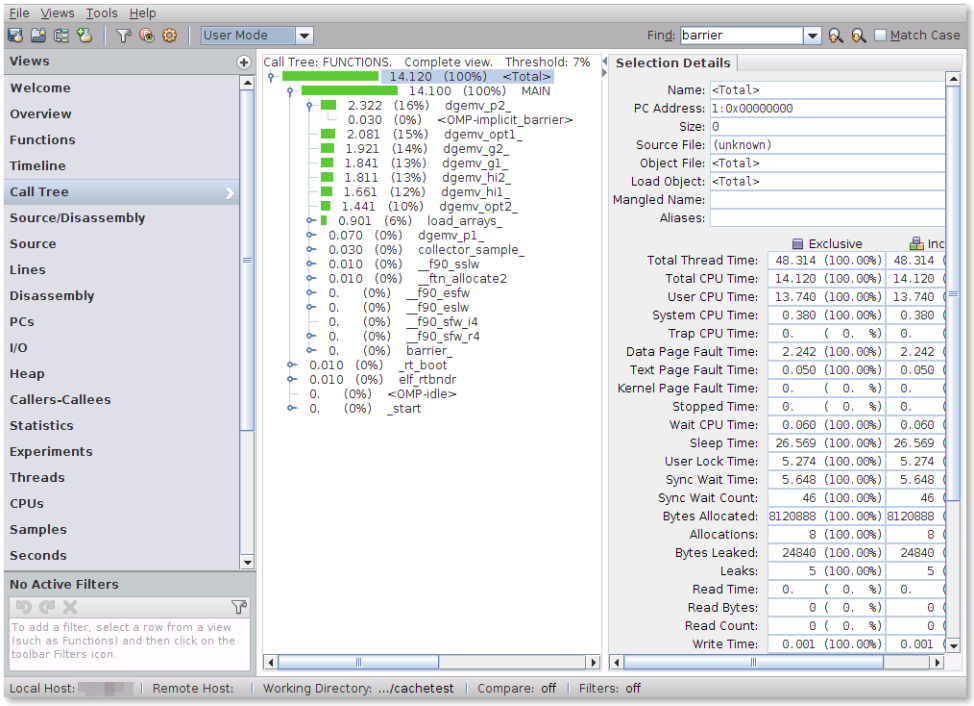
Call Tree Improvements

Call Tree view features the following enhancements:

- Metric percentages are shown in color bars.
- New Threshold setting enables you to specify when to expand branches with high metrics.
- Show Next Reference and Show All References context menu items enable you to find branches that include a selected function.
- New actions are provided in the right-click menu for showing and hiding tooltips and color bars.
- A function you select in another data view is reflected in Call Tree by expanding the hottest branch that contains the function, and a function you select in Call Tree is also selected in other views.
- You can sort by name or metric.

See the Help in the Performance Analyzer for more information about the Call Tree view.

FIGURE 3-6 Call Tree View



Java Profiling Improvements

Profiling of Java application is improved in the following ways:

- Java profiling is now enabled by default whenever the target application is a Java virtual machine. Previously you needed to specify a `-j on` option with the `collect` command to profile a Java application.
- You can now profile a running Java application on Oracle Solaris using the Profile Running Process dialog in the Performance Analyzer. The Collector attaches to the process and records profiling data and call stacks for the JVM and the Java program.
- Performance Analyzer and `er_print` can show Java thread names and thread groups and allows you to use them for filtering.
- Profiling of JDK 8 is now supported.

See the Java profiling tutorial Chapter 3, “Introduction to Java Profiling,” in “Oracle Solaris Studio 12.4: Performance Analyzer Tutorials ”.

Simplified Hardware Counter Profiling

For Oracle hardware, default profiling can now be enabled with a new option, "`-h on`". This option will select counters that measure CPI/IPC and high-latency memory accesses.

Specification of hardware counters has also been simplified. On Solaris, [`on`|`high`|`low`] now selects profiling rates that approximate the rates used for clock profiling. The [`on`|`high`|`low`] options take most of the guesswork out of choosing appropriate overflow periods, greatly reducing the risk of extreme under- and over-sampling. On Linux, the [`on`|`high`|low] options are supported for aliased hardware counters, but numeric specifications are required for raw counters.

For detailed information about how to use hardware counter profiling see Chapter 5, “Hardware Counter Profiling on a Multithreaded Program,” in “Oracle Solaris Studio 12.4: Performance Analyzer Tutorials ”.

Memoryspace Profiling Improvements

Memoryspace profiling enables you to see which memory addresses are costing the most performance. Memoryspace profiling is available on SPARC platforms running Oracle Solaris 10 and 11, and on x86 platforms running Oracle Solaris 11.2.

This type of profiling makes use of hardware counters known as *precise load-store* counters.

Use the command `collect -h` to see which precise load-store counters are available on your system. See the `collect`(1) man page for more information about how to perform memoryspace profiling using these counters.

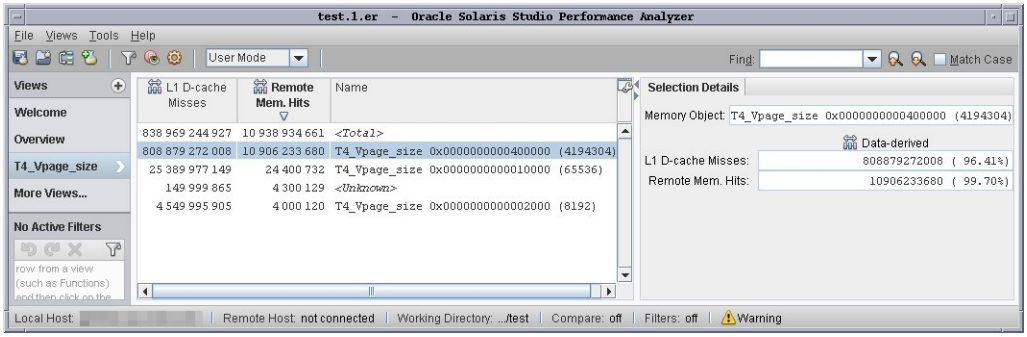
Improvements in memoryspace profiling include:

- Hardware counter profiling using the default `-h on` option typically includes at least one memoryspace counter
- You no longer need to use a + sign with a precise counter to trigger memoryspace profiling
- Memoryspace profiling data views are available if data is present in the experiment

When you open a memoryspace profiling experiment in Performance Analyzer you must enable the relevant hardware counter in the Overview page or Settings dialog to display the counter in data views. Choose a memoryspace view from the Views menu.

A sample data view of cache misses by memory page is shown in the following figure.

FIGURE 3-7 Memory Page View Showing Memoryspace Profiling Metrics



Performance costs can be attributed to cache lines and memory pages. When used with filtering, this data can identify exactly which source code lines are making specific, high-latency memory references.

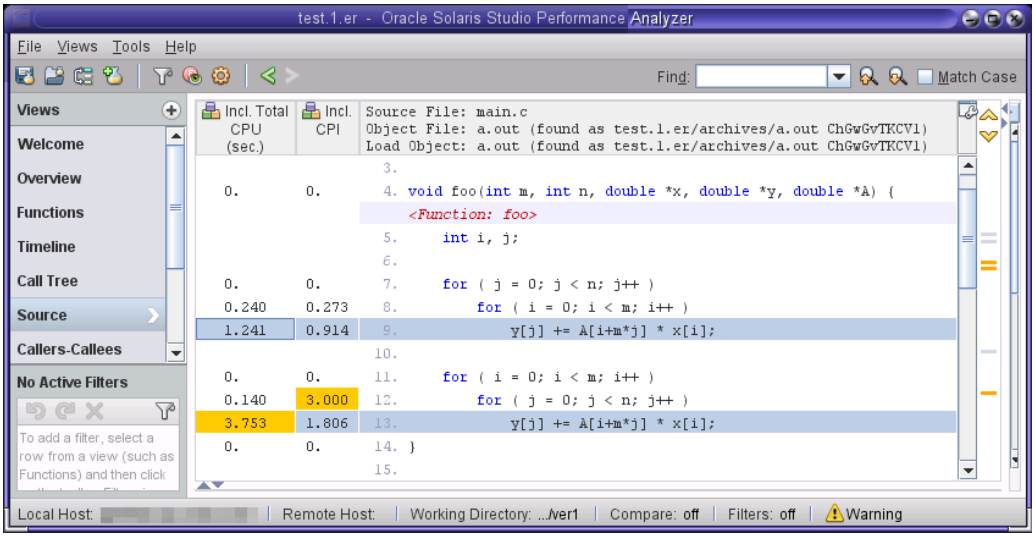
New Derived Metrics: CPI and IPC

Performance Analyzer displays new derived metrics called CPI (Cycles Per Instruction) and IPC (Instructions Per Cycle) which can help you identify where your application is running efficiently or inefficiently. The CPI and IPC metrics are available when you perform hardware counter profiling on your application and specify the counters for cycles and instructions. For example, you could profile using the command `collect -h cycles,on,insts,on` to generate the metrics.

Inefficient areas display with a high CPI or low IPC. Efficient areas of your program show low CPI or high IPC.

The following figure shows the Source view with a high CPI metric on line 12, which indicates it is running inefficiently.

FIGURE 3-8 CPI Metric in the Source View



Cross-Platform Analysis

Performance Analyzer can read experiments that were recorded on any supported OS and architecture, independent of the OS and architecture of the system on which it is currently running. When it runs on a remote client system, Performance Analyzer can read experiments recorded on any supported OS and architecture, independent of the remote host.

For example, if you are running Performance Analyzer remotely on a Windows laptop, you can connect to a remote x86 based system running Oracle Solaris and open an experiment that was recorded on a SPARC based system running Oracle Solaris.

Remote Use of Performance Analyzer

You can connect to a remote host from the Performance Analyzer to examine experiments on a remote system. When you use a remote connection you have the advantage of examining experiments in the same environment where they were recorded.

Requirements of the client system:

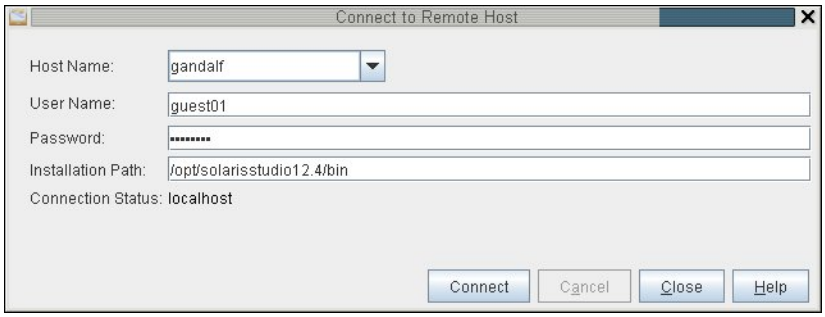
- The client operating system must be Mac OS X, Windows, Linux, or Oracle Solaris.
- Java 1.7 must be in your path on the client system.
- The version of the Performance Analyzer installed on the client must match the version of Oracle Solaris Studio tools installed on the remote system.

Requirements of the remote host:

- The remote host operating system must be Oracle Solaris *or Linux*.
- The remote host must run the Secure Shell (SSH) daemon, sshd.
- Oracle Solaris Studio 12.4 software must be accessible on the remote host and you need to know the path to the tools.
- You must have a user account on the host.

On Linux and Oracle Solaris systems where Oracle Solaris Studio is installed, you can connect to remote systems by choosing File > Connect To Remote Host to open the Connect to Remote Host Dialog shown in the following figure.

FIGURE 3-9 Connect to Remote Host Dialog



For use on systems where this release of Oracle Solaris Studio is not installed, a client distribution of Performance Analyzer is provided as a `RemoteAnalyzer.tar` file in the `lib/analyzer` directory in the full Oracle Solaris Studio installation. To install the client distribution of Performance Analyzer, copy the tar file to your system and unpack it. The directory `RemoteAnalyzer` contains scripts for Windows, Mac OS, Linux, and Solaris, and a `README.txt` file describing the functionality and usage, and a `lib` subdirectory containing the required components for remote operation.

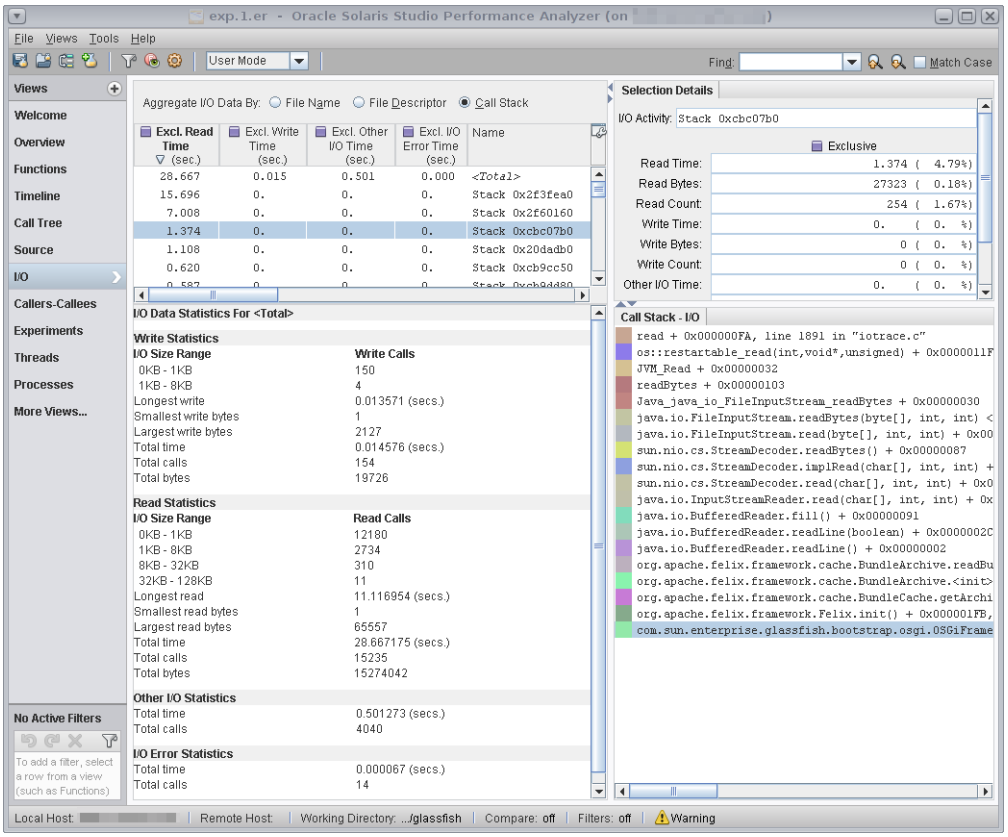
You start Performance Analyzer on the client system by executing the script that is appropriate for your system. When Performance Analyzer starts, it shows the Welcome screen with only those features that work remotely enabled. For more information, see the `RemoteAnalyzer/README.txt` file and the Help menu in Performance Analyzer.

New I/O Data View

A new I/O view can help you identify the I/O patterns in your application and pinpoint the I/O bottlenecks that impact its performance. The I/O view is available if you profiled your application for I/O tracing data from Performance Analyzer, or using `collect -i on` command.

I/O view shows read and write data aggregated by file name, file descriptor, or call stack. You can also use it to filter I/O events from your data. A new Duration view is available for analyzing the time duration of I/O operations. A new Data Size view is available that shows the distribution of I/O operations by byte size. See the Help in the Performance Analyzer for more information about the I/O view.

FIGURE 3-10 I/O Data View

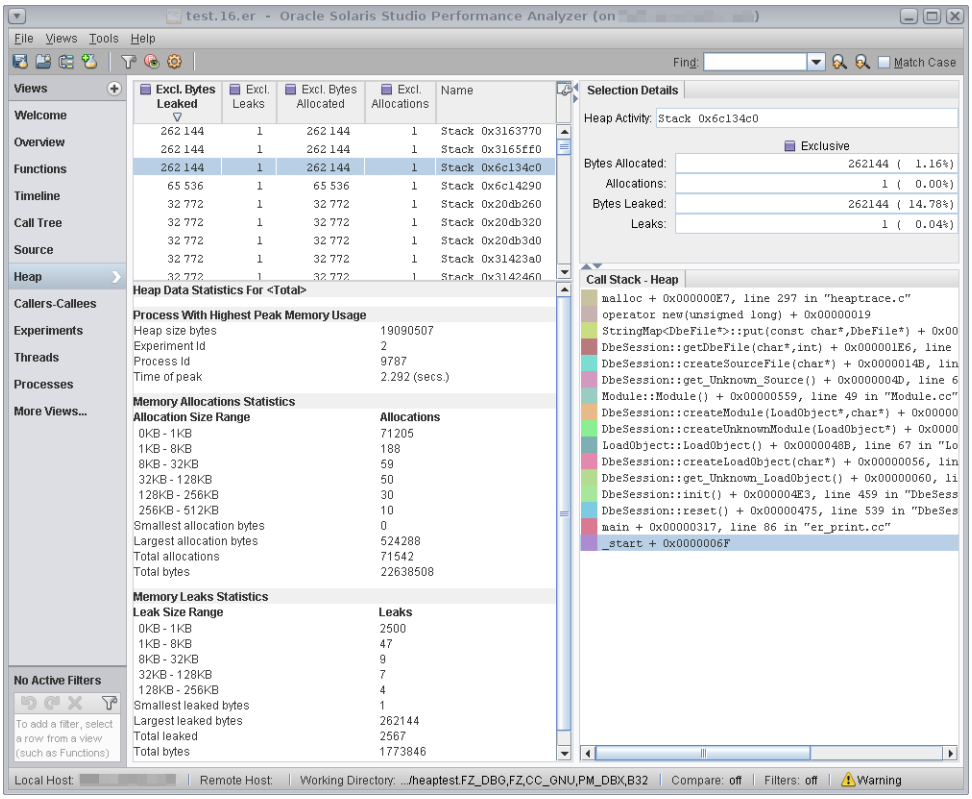


New Heap Data View

A new Heap view shows possible memory leaks in your program. The view is available if you profiled your application for heap tracing data from Performance Analyzer, or using the `collect -H on` command.

The Heap view shows a list of call stacks that have memory allocation metrics which indicate possible memory leaks. Timeline now shows the size of the allocated heap as a function of time. A new Data Size view is available that shows the distribution of allocations and leaks by number of bytes. A new Duration view is available for analyzing the duration of allocations. See the Help in the Performance Analyzer for more information about the Heap view.

FIGURE 3-11 New Heap Data View



Other Changes to Performance Analyzer

The Performance Analyzer tool features the following other enhancements.

- You can now specify a path in which to store Performance Analyzer's user settings by using the `analyzer` command's `-u` or `--userdir` argument.
- Comparing and aggregating experiments is now easier to do. You can choose between absolute values or deltas to show the change in metric values when comparing experiments.
- Aggregation of performance metrics by shared library or Java class is greatly improved, and the dialog supporting it has been renamed from "Show/Hide/API-only" to "Set Library and Class Visibility
- The Print option was replaced by Export to a file and supports more outputs. In some views, you can now export to an ASCII text table, a delimiter-separated list, or an HTML table. All exports are to the file system on the machine on which the Performance Analyzer is running. If you are running in remote mode, you export the data to a file on the remote machine.
- The names of the metrics for clock profiling, MPI tracing, and heap tracing have changed. For details, see the `collect` (1) man page.
- Performance Analyzer shows Java thread names and thread groups, and you can use them for filtering.
- Performance Analyzer now supports attaching to a running process and profiling it. Choose File > Profile Running Process or click Profile Running Process on the Welcome screen. Previously you could only attach to a running process by using the `collect` command or `dbx collector` command.
- You can now profile the Oracle Solaris kernel from Performance Analyzer. Choose File > Profile Kernel or click Profile Kernel on the Welcome screen. Previously you could only profile the kernel by using the `er_kernel` command.
- Performance Analyzer has a new way to save settings for persistence. When you exit Performance Analyzer, most settings such as metrics and views are persistent, so that the next time you open the same experiment it is displayed the same way as when you last closed it. You can save selected settings in a configuration file and apply the configuration to the same or different experiments when you open them from the Open Experiment dialog box. Performance Analyzer can also save settings into a `.er.rc` file to be read by `er_print`.

Changes to Command-Line Tools

This section describes changes made to various command-line performance analysis tools.

Changes to Data Collection Tools

Data collection tools include the `collect` command, `dbx collector` command, and `er_kernel` command. Each of these tools is used to profile programs to collect data and create experiments that can be read by Performance Analyzer or `er_print`.

The following changes are common to these tools:

- New processors are supported for hardware counters and stack unwind: SPARC T5, SPARC M5, SPARC M6, SPARC 64 X, SPARC 64 X+, Intel Ivy Bridge and Haswell.
- Clock profiling is enabled by default even if hardware counter profiling is not specified.
- Setting archive on is the same as setting archive copy. For `collect` and `er_kernel` this means `-A on` is now the same as `-A copy` . For `dbx collector` this means `collector archive on` is the same as `collector archive copy`.
- The maximum number of threads on Oracle Solaris that can be profiled is now 32768.

`collect` Utility Changes

The `collect` utility is a tool you use to profile your application as it runs to collect data and create an experiment that can be read by Performance Analyzer or `er_print`.

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

- I/O tracing is supported with a new `-i` flag.
- Java profiling is enabled by default whenever the target is a JVM. You no longer need to specify `-j on`.
- `-P` option for collecting data from a running process is now supported on Linux systems. Note that this only works for single-threaded applications.
- The `-c` option for collecting count data is now supported on Linux.
- Hardware counter processing now supports multiple `-h` arguments and a default counter set. You can set the environment variable `SP_COLLECTOR_HWC_DEFAULT` to enable hardware counter profiling by default.
- Hardware counter-based memoryspace profiling is enabled by default for precise counters on SPARC and x86 systems. The plus sign (+) is no longer required to capture memory addresses. See “Memoryspace Profiling Improvements” on page 28 for more information.
- `collect -F =expr` will no longer match the expression against the process lineage.
- When using `-P` to attach to a Java program, be sure to specify `-j on`.

`dbx collector` Changes

The `dbx collector` is a subcommand of the `dbx` debugger that you can use for performance data collection. See the `collector`(1) man page for more information.

In addition to the changes common to all data collection tools the `dbx collector` command is changed in this release as follows:

- Another experiment can be started after a `detach` or experiment termination.
- Attach is supported on Linux system for single-threaded native applications. For more information about limitations with attaching, see "dbx attach Profiling (collect -P)" in "Oracle Solaris Studio 12.4: Release Notes ".
- `dbx collector` supports the following new commands:
  - `collector iotrace on` - Specifies to turn on I/O tracing.
  - `collector duration` - Specifies a time range for running an experiment.
  - `collector java` - Specifies whether to collect Java profile data. Default is `off`. You should use the command whenever profiling Java, whether by attaching or launching the JVM.
  - `collector pausesig` - Specifies a signal to be used for pausing or resuming data collection.
  - `collector samplesig` - Specifies a signal to be used to record a sample.
  - `collector hwprofile addcounter` - Specifies additional counters for hardware counter overflow profiling.

`er_kernel` Utility Changes

The `er_kernel` command profiles the Oracle Solaris kernel and generates an experiment that you can examine in Performance Analyzer or `er_print`.

In addition to the changes common to all data collection tools, the `er_kernel` utility is changed as follows:

- Clock-profiling metrics in user sub-experiments are recorded as in `collect` experiments, but only User CPU Time and System CPU Time are recorded; wait times are not recorded.
- The names of hardware counter metrics reported in the kernel founder experiment are prefaced by `K_` . Metrics in user sub-experiments use the same names as metrics in `collect` experiments.
- Kernel dataspace profiling is supported on Solaris SPARC, for precise counters, and for systems with version 1.8 of DTrace, or later.
- Clock-profiling for the founder experiment is now recorded as in `collect` experiments, except they have only a Kernel CPU Time metric. Hardware counter profiling is not enabled by default.

- Improved recording of kernel call stacks.

`er_print` Utility Changes

The `er_print` utility generates a plain-text version of the data views presented by the Performance Analyzer. The output is displayed on the standard output.

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

- IO tracing data is available in three new reports: `ioactivity`, `iodetail`, and `iocallstack`. In the `ioactivity` report, data is aggregated by file name. In the `iodetail` report, data is separated by each file descriptor as it is opened. In the `iocallstack`, data is aggregated by common call stack.
- Heap tracing data is available in new reports: `heap` and `heapstat`. Use `heap` to print all allocations and leaks aggregated by common callstack. Use `heapstat` to print a summary of heap usage statistics including peak usage over the life of the process.
- New derived metrics CPI (cycles per instruction) and IPC (instructions per cycle) are available when you profile your application to collect `cycles` and `insts` hardware counters. `CPI` and `IPC` can be specified as metrics.
- A new report, `overview` , shows summary information from the experiments.
- The presentation from `object_list` now includes the index (as shown in PCs in other data views), the full path to the object, and its setting for visibility of shared-library functions.
- `er_print` can now read experiments that were recorded on any architecture while it runs on any other supported architecture.
- A new `printmode` *string* command, supports printing either ASCII form, as in previous releases (`string` = `table`), or in a delimiter-separated list (`string` = *X* , with X as any single-character delimiter), or as a table in HTML format (`string` = `html`). A `printmode` command can be in an `er.rc` file.
- Memory Objects specific for a machine will be created if a `machinemodel` is specified during user `er.rc` processing, or as an input command, or as a command in a script, or if an experiment being loaded had recorded a `machinemodel` .
- `er_print` shows Java thread names and thread groups, and enable filtering by them.
- Comparison mode supports displaying compared data as absolute values or deltas to show the change in metric values between experiments.
- `setpath` directives are no longer permitted in `.er_rc` files. If you have `setpath` directives in `.er_rc`files that you created in a previous release, you should change the `setpath` lines to `addpath`.

Changes to Other Commands

`er_archive` command is changed in this release as follows:

- `er_archive` (and all other tools) can process executables whose symbolic information is recorded in a side file.
- `er_archive` now only copies shared objects, and does not generate an archive file; the `-A` flag is silently ignored.
- `er_archive` supports a `-s` *type* flag to specify archiving or source, object, and ancillary files. if *type* is `all`, all source files that can be found are archived; if *type* is `used` only those source files that are referenced in the experiments are archived.
- `er_archive` examines an environment variable, `SP_ARCHIVE_ARGS`, which you can use to specify `-s` and `-m` arguments. If the `SP_ARCHIVE_ARGS` variable is set when an experiment is recorded, the `er_archive` that is run automatically can do source archiving. It will be processed when `er_archive` is run manually.

Changes to Experiments

Performance data experiments have changed as follows:

- The experiment format has changed and the version number is now 12.4.
- This release of the performance analysis tools can read the following experiment versions:

10.2	created with a pre-release Oracle Solaris Studio 12.3
12.3	created with the released Oracle Solaris Studio 12.3
12.4	created with this Oracle Solaris Studio 12.4

  If you try to use an experiment older than version 10.2, the tools display a message that the experiment must be read with an earlier version of the tools.
- Clock-profiling metrics have been reorganized to show more detail, some metrics have been renamed, and different defaults have been set. For more detail, see the `collect` man page, which now lists the metric abbreviations as well as the names.

♦ ♦ ♦ CHAPTER 4

Code Analysis Tools

The code analysis tool suite ensures application reliability and stability by detecting common coding errors, including memory leaks and access violations, enabling developers to write better code with fewer errors faster.

This chapter describes the new and changed features in the code analysis tools in this Oracle Solaris Studio release and contains the following sections:

About the Code Analysis Tools

The code analysis tools help you make your application more reliable by using static, dynamic, and code coverage analysis to detect many common coding errors, including memory leaks and memory access violations. Previse performs static analysis at compilation and Discover performs dynamic analysis at application runtime to identify code quality issues. The Uncover tool analyzes code coverage data to provide information about functions that are not covered by your test suite and tells how you can benefit by covering those functions.

Use the Code Analyzer graphics tool or the new `codean` command-line utility to view the three types of analysis to get a comprehensive view of your application's vulnerabilities so you can improve its correctness and reliability.

New Command-Line Code Analyzer Tool codean

A new utility called codean enables you to view the code analysis data generated by Previse, Discover, and Uncover using the command line instead of using the Code Analyzer graphical tool. The codean tool provides functionality similar to the Code Analyzer, but you can use it on systems where a graphical environment is not available, or if you prefer the command line. The codean tool can also be used in automated scripts and has some features that are not yet available in the Code Analyzer tool.

The codean tool has the following unique capabilities:

- See results in html and text formats.
- Save tool reports for comparison with new versions of the binary being checked.
- Compare the current report to saved reports to display only new errors. . See "Using the --whatisnew Option" on page 40.
- Compare the current report to saved reports to display only fixed errors. See "Using the --whatisfixed Option" on page 41.
- Show all errors from multiple dynamic error checking runs. For example, if a test suite is run with a Discover instrumented binary, aggregate all the data in one text report.
- Compile a summary html page for all reports under a directory. See "Using codean to Generate a Summary HTML Page" on page 41.

For more information about codean , see the codean(1) man page.

Using the --whatisnew Option

Use the --whatisnew option to generate a report of only the new errors by comparing against a previously saved tool report. For example, use the Code Analysis tools to create a frozen copy of the state of the source base at the time of the tools adoption. You could then use --whatisnew to ensure ongoing changes to the source base do not create any new security vulnerabilities.

The following is an example of using the --whatisnew to display only new errors:

```
%codean --whatisnew a.out
STATIC report of a.out showing new issues:
Compare the latest results against a.out.analyze/history/09:58:35May152013...
MEMORY LEAK 1 : 1 block left allocated on heap with a total size of 400 bytes
    sample1()  <sample1.c : 20>
        17:    {
        18:         global = (int *)malloc(100);
        19:         int *p = malloc(100*sizeof(int));
```

```
        20:=>         int *q = malloc(100*sizeof(int));
        22:         add_0_1_put_in_2(p);-
PREVISE SUMMARY:
    0 new error(s), 0 new warning(s), 1 new leak(s) in total
```

Using the --whatisfixed Option

The --whatisfixed option complements the --whatisnew option by generating a report that shows only the errors that occur in a previous cached tool report. Development or Quality Assurance teams could use this option to track security bug fixes.

The following is an example of using --whatisfixed to display only fixed errors:

```
% codean --whatisfixed a.out
STATIC report of a.out showing fixed issues:
Compare the latest results against a.out.analyze/history/10:02:05May152013...
MEMORY LEAK 1 : 1 block left allocated on heap with a total size of 400 bytes
(Warning: Source files have changed. Source code shown below may not be accurate.)
    sample1()  <sample1.c : 20>
        17:    {
        18:         global = (int *)malloc(100);
        19:         int *p = malloc(100*sizeof(int));
        20:=>         int *q = malloc(100*sizeof(int));
        21:         free(q);
PREVISE SUMMARY:
    0 fixed error(s), 0 fixed warning(s), 1 fixed leak(s) in total
```

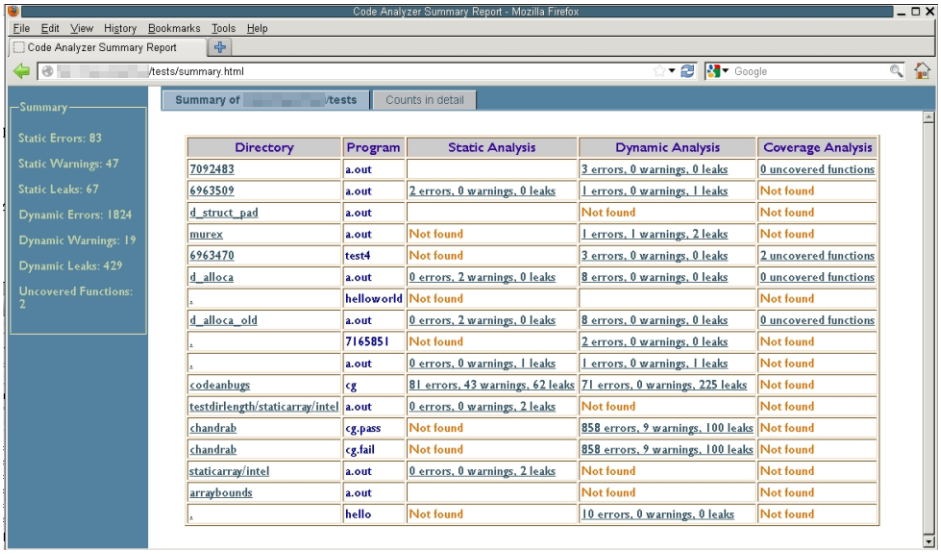
Using codean to Generate a Summary HTML Page

You can create a summary report of all information generated by the code analysis tools that you previously ran on multiple binaries in a directory tree. The HTML page organizes all errors from static, dynamic, and coverage reports for multiple binaries into a neat table.

For example, if you want to create an HTML page in your top directory ./tests, use codean ./tests.

The following figure is an example of a generated summary.html page with all the reports under a /tests directory.

FIGURE 4-1 Summary HTML Report



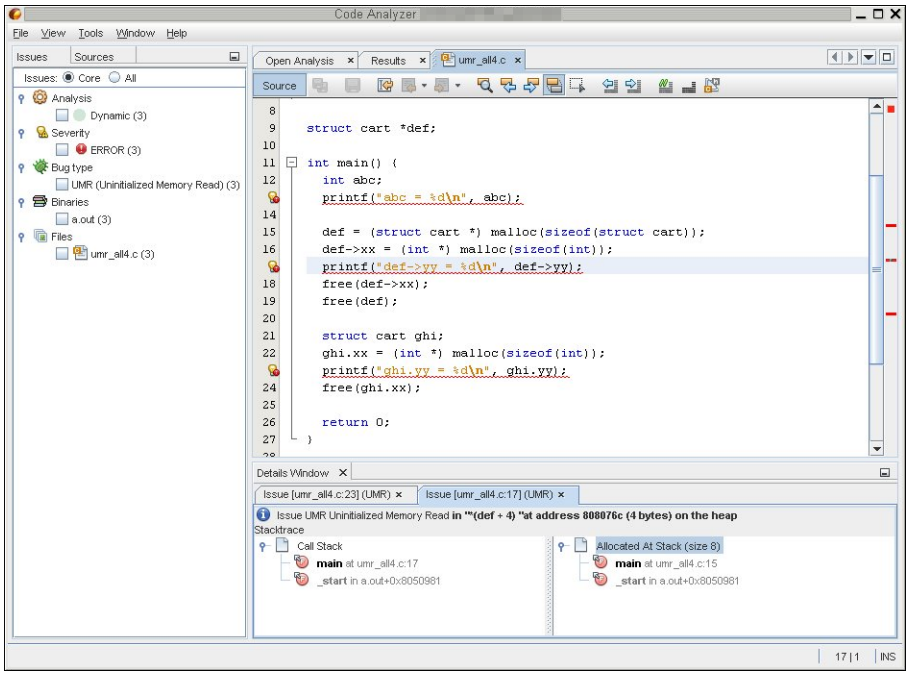
Code Analyzer Changes

The following common features were added to all Code Analyzer component tools. For more information, see the Help in the Code Analyzer and "Oracle Solaris Studio 12.4: Code Analyzer User's Guide ".

- Now available on Oracle Enterprise Linux..
- Improved enterprise application support by using 64-bit technology in the Code Analyzer components instrumentation engine.
- Significant improvements to the instrumentation engine runtime.
- Ability to instrument large applications on small machines due to big reductions in engine memory footprint.
- Support for SPARC T5 and M5 processors added.
- Support for Intel Ivy Bridge processors added.
- Added display variable name information in dynamic error reports.

The following figure shows the Code Analyzer GUI with a code example displaying variable names.

FIGURE 4-2 Code Analyzer GUI Screenshot with Variable Names



For more information about Code Analyzer in general, see the Help in Code Analyzer, the "Oracle Solaris Studio 12.4: Code Analyzer User's Guide ","Oracle Solaris Studio 12.4: Code Analyzer Tutorial ", and the `code-analyzer` (1) man page.

New Previse Static Analysis Features

The following features were added in this release to the Previse static analysis tool:

- To collect static code errors, you can now use the `-xprevise` option when compiling your code, instead of the `-xanalyze=code` option which is EOL.
- Static analysis is now available on Oracle Enterprise Linux.
- Significant improvement of accuracy by eliminating false positives.

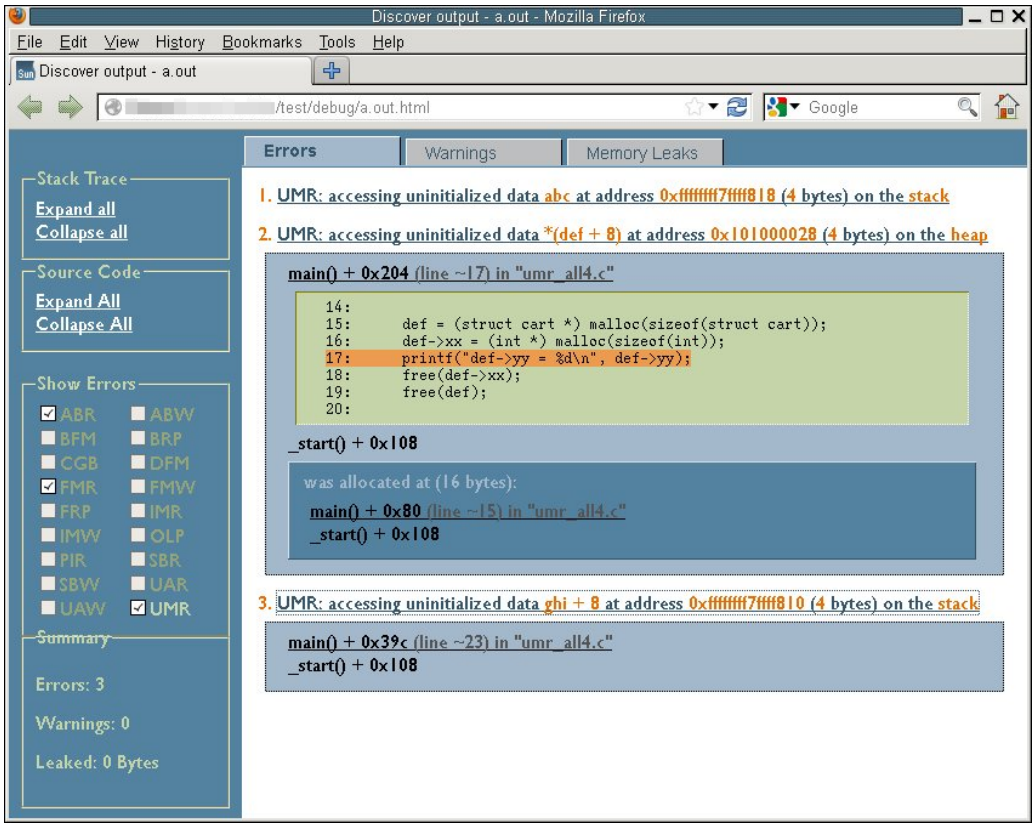
New Discover Features

The following features were added in the Discover memory analysis tool in this release. For more information, see the `discover` (1) man page and "Oracle Solaris Studio 12.4: Discover and Uncover User's Guide ".

- Discover can check parts of an executable or a library, by using the `-c[- | lib[:scope...] | file]` option. For more information, see "Checking Parts of a Library or an Executable" in "Oracle Solaris Studio 12.4: Discover and Uncover User's Guide ".
- New Discover APIs display memory leaks and memory usage on demand. For more information, see "New Discover APIs" on page 45.
- Discover is now available on Oracle Enterprise Linux.
- Discover can follow and collect memory access data from both the child and parent process with the`-F both` option. This is the new default.
- Discover error reports support multiple runs of the target binary as in a test suite. Error reporting format works in conjunction with the new command line `codean` utility.
- New functionality to check for memory errors on code allocated with `mmap(2)`.
- HTML reports improved to highlight variable names, line numbers, and addresses.
- Name of variable is now shown in memory error where the memory corruption occurs.
- Discover can now catch static-type array out-of bounds errors by default. For more information, see "Memory Access Errors and Warnings" in "Oracle Solaris Studio 12.4: Discover and Uncover User's Guide ".
- Using the `-i datarace` option can now report dual stack traces for races detected with binary instrumentation using Discover.
- Support added for large files.

The following figure is an example of using Discover to generate a report with HTML highlighting and variable name.

FIGURE 4-3 Discover HTML Highlighting and Variable Name Example



New Discover APIs

Six new Discover APIs were added to the Code Analysis tools. When your application is using too much memory, you can use these APIs to pinpoint where the memory leaks are happening before the application ends. The APIs can also locate all the memory blocks in use.

These APIs can be inserted directly to the application source or called during a debugging session to report the new and total memory usage, and the new and total memory leaks, at any time, from any start point. They are especially useful for long-running or always-running enterprise server applications.

The following APIs were added to the Code Analysis tools:

- `discover_report_all_inuse()`
- `discover_mark_all_inuse_as_reported()`
- `discover_report_unreported_inuse()`
- `discover_report_all_leaks()`
- `discover_mark_all_leaks_as_reported()`
- `discover_report_unreported_leaks()`

EXAMPLE 4-1 Using the `discover_report_unreported_leaks()` API

The following is an example of using the `discover_report_unreported_leaks()` API, after running the binary with Discover and using `dbx` to execute the API.

```
%discover -w - a.out
% dbx a.out
 (dbx) stop in foo2
 (dbx) run
 (dbx) call discover_report_unreported_leaks()
 ******** discover_report_unreported_leaks() Report ********

  1 allocation at 1 location left on the heap with a total size of 1 byte

      LEAK 1: 1 allocation with total size of 1 byte
          foo1() + 0x5e  <api_example.c:7>
                    4:    #include <discoverAPI.h>
                    5:
                    6:    void foo1() {
                    7:=>    char *x = (char *) malloc(sizeof(char));
                    8:      *x = 'a';
                    9:      printf("x = %c\n", *x);
                   10:      /*free(x);*/
          main() + 0x1a  <api_example.c:21>
                   18:    }
                   19:
                   20:    int main() {
                   21:=>    foo1();
                   22:      foo2();
                   23:      return 0;
                   24:    }
          _start() + 0x71

  ********************************************************
```

For more information on each of the Discover APIs and how to use them, see the Discover header file and “discover APIs” in “Oracle Solaris Studio 12.4: Discover and Uncover User’s Guide ”.

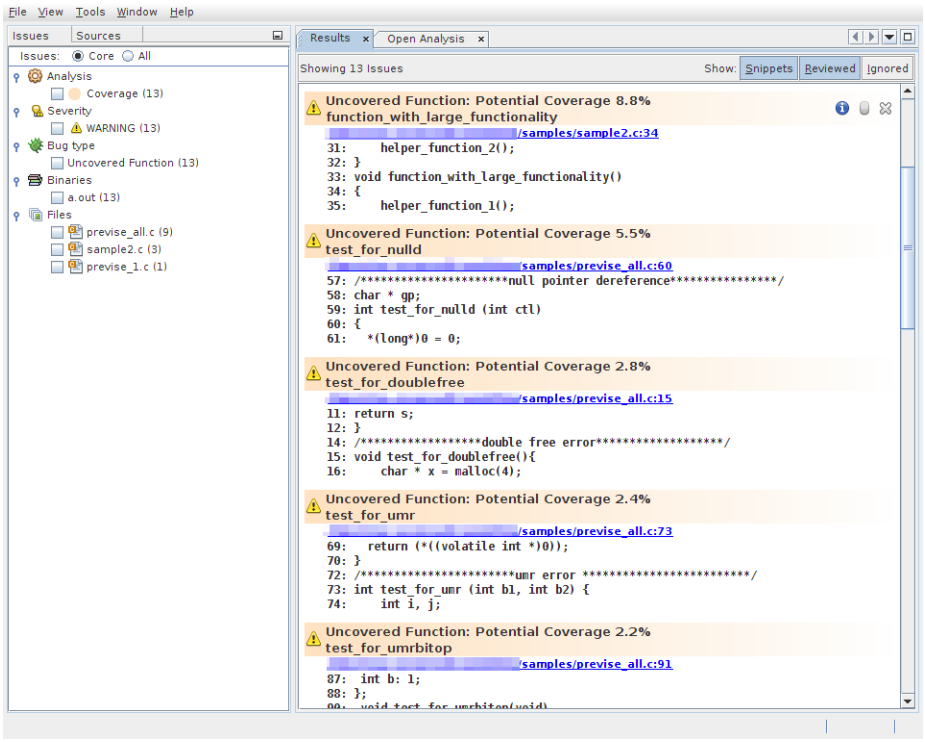
New Uncover Features

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

- Uncover is now available on Oracle Enterprise Linux.
- Improved correctness in coverage reports for C++ applications.
- Support for unusual process termination usually seen in long running server applications.
- Reduced time and memory footprint overhead in Uncover instrumentation and runtime.
- Uncover can now get coverage and profiling data while the program is running.
- Support added for large files.

The following figure is an example of using Uncover with the Code Analyzer GUI:

FIGURE 4-4 Example of Uncover Results in Code Analyzer GUI



For more information, see the `uncover` (1) man page and the "Oracle Solaris Studio 12.4: Discover and Uncover User's Guide ".

♦ ♦ ♦ CHAPTER 5

Debugging Tools

Oracle Solaris Studio provides the command-line `dbx` debugger, and the `dbxtool` graphical tool for using `dbx`. The debugger is also integrated into the IDE. For more information about debugging with the IDE, see Chapter 6, “Oracle Solaris Studio IDE”.

This chapter contains the following topics about what's new in the debugging tools:

About the `dbx` Debugger

The `dbx` debugger is an interactive, source-level, postmortem and real-time debugging tool. You can use it at the command line, through the `dbxtool` graphical interface, and in the Oracle Solaris Studio IDE. The `dbx` debugger is scriptable and multithread-aware.

New and Changed `dbx` Features

The following features were added or changed in `dbx` . For more information, see “Oracle Solaris Studio 12.4: Debugging a Program With dbx ”, the `dbx` (1) man page and the `dbx` help file.

- Start-up time for `dbx` reduced drastically, with an up to 7x improvement for enterprise applications.
- New compiler and linker options for `dbx`. See “New Compiler and Linker Options to Support Debugging” on page 50 for more information.
- New embedded Python interpreter, which enables you to write pretty-printing filters for C and C++ expressions on Oracle Solaris. See “Pretty-Printing With Python” on page 52 for more information.

- New dbxenv variable output_pretty_print_mode that determines which pretty-printing mechanism is used. If set to call. uses call-style pretty-printers. If set to filter, uses Python-based pretty-printers. If set to filter_unless_call, uses call-style pretty-printers first.
- New dbxenv variable filter_max_length. Set this dbxenv variable to the maximum length of sequences converted to arrays by pretty-printing filters.
- Support for the C++11 standard added.
- Support for the C11 standard added.
- Support for the following compiler options added. For more information about these options, see: "Changes to Compilers" on page 69.
  - -g1
  - -xdebuginfo
  - -xglobalize
  - -xpatchpadding

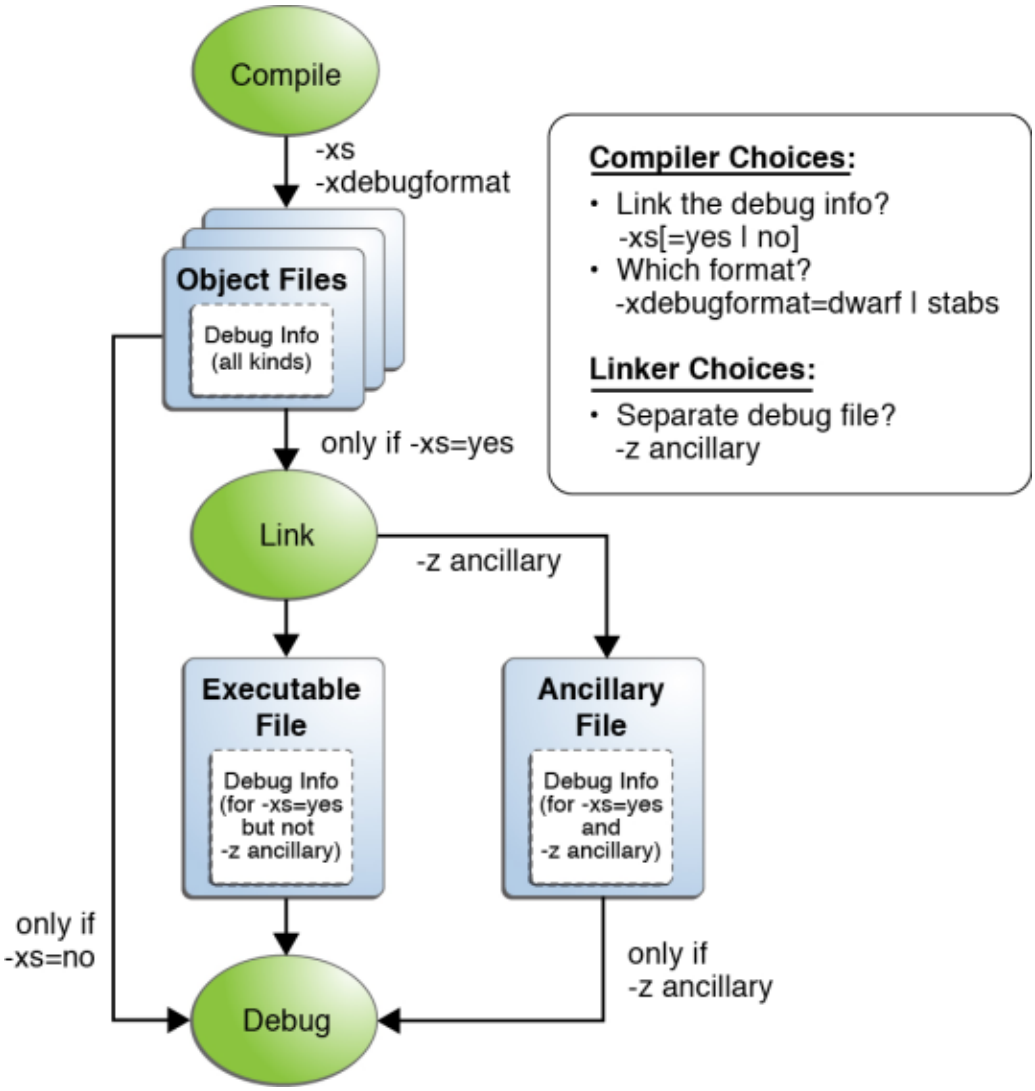
For more information, issue a help changes command under dbx, to access the dbx help file.

New Compiler and Linker Options to Support Debugging

New compiler and linker options give users more freedom to generate and use debug information. Compilers generate an Index for DWARF, similar to index stabs The index is always present and results in faster dbx start-up time, as well as other improvements when debugging with DWARF.

The following is a diagram of the different kinds and locations of debug information, specifically highlighting where the debug data resides:

FIGURE 5-1 Flow of Debug Information



Index DWARF (`-xs[={yes|no}]`)

DWARF by default is loaded into the executable file. The new index makes it possible to leave the DWARF in the object files with the `-xs=no` option. This results in a smaller executable size

and a faster link. The object files must be retained in order to debug. This is similar to how stabs works.

Separate Debug File (-z ancillary[=*outfile*])

The Oracle Solaris 11.1 linker can send debug information to a separate ancillary file while building the executable. A separate debug file is useful for environments where all the debug information must be moved, installed, or archived. An executable can be run independently, but can also be debugged by people with a copy of its separate debug file.

dbx continues to support the use of the GNU utility objcopy to extract debug information into a separate file, but using the Oracle Solaris linker has the following advantages over objcopy:

- The separate debug file is produced as a by-product of the link
- A program which was too large to be linked as one file links as two files

Note - Using both the Index DWARF and a separate debug file has the effect of copying just the index, which carried only the information being linked, into the separate debug file. This is typically not very useful, unless that small amount of space used by the index is critical.

Minimizing Debug Information

The -g1 compiler option is intended for minimal debuggability of deployed applications. Compiling your application with this option produces the file and line number, as well as simple parameter information that is considered crucial during postmortem debugging. For more information, see the compiler man pages and the compiler user guides.

Pretty-Printing With Python

dbx now has a mechanism in which you can write pretty-printing filters in Python. The pretty-printing filters transform a Value to a more readable form in dbx.

On the dbx command line, you can enable pretty-printing by using the -p option for the print, display, and watch commands or by typing dbxenv output_pretty_print on. In the IDE and dbxtool, you can enable pretty-printing by setting your dbxenv variable output_pretty_print to on and you can use the Pretty Print checkbox in the context menu of the Watches and Variables windows.

Filters are built in for select classes in 4 implementations of the C++ Standard Template Library. The following table specifies the library name and the compiler option for that library:

Compiler option for Library	Library Name
`-library=Cstd` (default)	`libCstd.so.1`
`-library=stlport4`	`libstlport.so.1`
`-library=stdcxx4`	`libstdcxx4.so.4.**`
`-library=stdcpp` (default when using the `-std=c++11` option)	`libstdc++.so.6.*`

The following table specifies which classes filters can be used for in the C++ Standard Template Library and if index and slice can be printed:

Classes	Index and Slice Available
string	N/A
pair	N/A
vector	yes
list	yes
set	yes
deque	yes
bitset	yes
map	yes
stack	yes
priority_queue	yes
multimap	yes
multiset	yes
tuple (C++ only)	N/A
unique_ptr (C++ only)	N/A

EXAMPLE 5-1 Pretty-Printing with Filters

The following output is an example of printing a list using the `print` command in `dbx`:

```
(dbx) print list10
list10 = {
   __buffer_size = 32U
   __buffer_list = {
     __data_ = 0x654a8
   }
   __free_list   = (nil)
   __next_avail  = 0x67334
   __last     = 0x67448
   __node     = 0x48830
   __length   = 10U
   }
```

The following is the same list printed in dbx, but using pretty-printing filters:

```
(dbx) print -p list10
list10 = (200, 201, 202, 203, 204, 205, 206, 207, 208, 209)

(dbx) print -p list10[5]
list10[5] = 205

(dbx) print -p list10[1..100:2]
list10[1..100:2] =
[1] = 202
[3] = 204
[5] = 206
[7] = 208
```

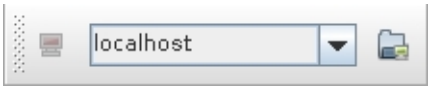
For general information about pretty-printing and call-style pretty-printers see "Using Pretty-Printing" in "Oracle Solaris Studio 12.4: Debugging a Program With dbx " and the topic `prettyprint` in the `dbx` helpfile.

dbxtool Changes

dbxtool is the stand-alone debugger GUI. All the functionalities of dbxtool provided in Oracle Solaris Studio 12.3 remain, but there is a new look-and-feel to dbxtool that is similar to Oracle Solaris Studio IDE.

The following features have been added to dbxtool:

- **Debug Recent Button** - dbxtool now has a Debug Recent button with a drop-down debug history list, enabling you to choose a different run of your code with different arguments. Pressing the Debug Recent button without choosing a specific run will debug the most recent target and arguments.
- **Code Assistance** - dbxtool has code-assistance automatically enabled in the editor while you are debugging your code. While you still cannot recompile your code directly in dbxtool, you can fix your code with the help of code assistance in the editor. To disable code assistance, right-click the debug target in the Projects tab and deselect the Code Assistance option. Alternatively, you can launch dbxtool with code-assistance turned off. For more information about code assistance, see "Code Assistance Improvements" on page 61.
- `--disable-code-assistance` **Option** - When launching dbxtool, you can specify the `--disable-code-assistance` option, if you do not want code assistance or if you have problems with memory usage and do not need code assistance.
- **Remote Host Toolbar** - You can now choose to manage and add remote hosts in dbxtool using the Remote Host Toolbar, which is enabled by default.



- **Debug Targets in Projects Tab** - dbxtool now automatically displays all of your debug targets in one place. Once you start a debugging session, the debug target is created and displayed in the Projects tab. The target information is stored in your `userdir` directory and persist between dbxtool runs. Expanding the debug target displays the folders and files from the root source directory, which are taken from the binary that was built.

♦ ♦ ♦ CHAPTER 6

Oracle Solaris Studio IDE

Oracle Solaris Studio IDE integrates many of the components of Oracle Solaris Studio for users who prefer a graphical programming environment.

The following topics are covered in this chapter:

About Oracle Solaris Studio IDE

Oracle Solaris Studio offers a graphical integrated development environment (IDE) that is built on the NetBeans platform and is configured to use the Oracle Solaris Studio C, C++, and Fortran compilers, the `dmake` distributed make command, and `dbx` debugger. The IDE also integrates with some of the Analyzer tools of the analysis suite so you can analyze your code without leaving the IDE.

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

For complete documentation of the IDE, see the Help in the IDE. For step-by-step instructions to use the basic features of the IDE, see the "Oracle Solaris Studio 12.4: IDE Quick Start Tutorial ".

New and Changed IDE Features

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

- **Debug executable binaries that are not in an IDE project (Projectless Debugging).** You can debug an executable by choosing Debug > Debug Executable and specifying the path to the executable and any arguments or environment variables needed to run it. You can also navigate to the executable file in the Favorites window, right-click the file and select Debug. The executable does not need to be part of an IDE project, but it should be located with the source code that was used to build it so the debugger can find debugging information.
- **C++11 support.** If your code uses the C++11 standard and your C++ compiler has implemented the C++11 standard, you can enable C++11 support in the IDE. This enables you to use code assistance for features such as the "auto" specifier, for example. To enable C ++11 support for a project, right-click the project, select Properties and then select Build > C++ Compiler > C++ Standard > C++11. To enable C++ support only on an individual file, right-click it, select Properties and then select C++ Compiler > C++ Standard > C++11.
- **Memory usage improvements.** The memory usage for big projects has been reduced by 50%.
- **Faster searching.** Find Usages is much faster and has an improved interface. Find Usages now runs in the background so you can do other tasks while it is searching a large number of files. The results are displayed immediately in a separate Usages panel and continues to display search results as they are incrementally found. The Usages panel provides a progress indicator and an incrementing count of occurrences. You can stop the search at any point and your search results up to that point are saved. You can navigate between the search hits, change the view from logical to physical, and run Find Usages again with different settings. Also, you can add filters to your searches and search in comments.
- **Lightweight partial reparse.** Editing a source file with a large set of dependencies is now much faster because of improvements in reparsing. Changes such as reformatting, editing inside function bodies, and adding comments or spaces no longer cause the full project to be reparsed. Certain changes cause the IDE to reparse the whole project.
- **Debugger breakpoint groups.** You can group breakpoints using several different categories, such as per file, per project, by type, by language, etc. In the Window > Debugging > Breakpoints window, click the Select breakpoint groups icon, and select the breakpoint group. The breakpoints are arranged according to your selection.
- **Window management and grouping.** You can perform actions on groups of windows in addition to individual windows. Each window belongs to a group that you can minimize/ restore, drag to a new location, float in a separate window, or dock back into the IDE window. For example the Projects, Files, Classes, and Services windows in the top left can be minimized by clicking the Minimize Window Group button on the right side of the group. Right-click in the tab area of the group to select an option for the group, or choose Window > Configure Windows. Options for controlling window behavior are in Tools > Options > Appearance > Windows.
- **Copy file path to clipboard.** You can hover the mouse cursor over any file in the IDE and press Alt+Shift+L to copy the file's path to the clipboard.
- **Launchers feature.** See “New Launchers Feature in IDE” on page 59.
- **Improvements in code editor.** See “IDE Code Editor Improvements” on page 60.
- **Improvements in code assistance.** See “Code Assistance Improvements” on page 61.
- **Breadcrumbs.** See “Using Breadcrumbs Navigation” on page 63.

- **The IDE's Action Items window now works with C/C++ projects.** The Action Items window shows lines in your source files that you've marked with comments like `TODO` or `FIXME` or `Pending` or `<<<<<<<<`.The strings detected are specified in Tools > Options > Team > Action Items. For C/C++ projects the Action Items window also shows compile errors and warnings that occurred when you last built the project.
- **Improved support for version control systems**:
  - Git repository support.
  - Local history : revert deleted and new History tab.
  - Shelve changes: allows you to put your local changes aside (shelve) and start working on a different feature. For Mercurial and Subversion, see Team > Shelve Changes menu option.
  - Mercurial enhancements: basic support for branches and tags and queues.
- **New solstudio command line options.**You can open and close project groups as the IDE starts using the `--open-group` and `--close-group`. options.
- **Bookmarks update.** Bookmarks view can be opened using Window > IDE Tools > Bookmarks and you can create bookmarks using Ctrl-Shift-M in your files.
- **Auto-completion in Search bar by pressing Ctrl-Space.** When you are searching in the editor, you can auto-complete your search term, the same as you would in get in the editor.
- **The concept of the "main project" is no longer used for most tasks.** You can still set a main project using Run > Set Main Project.
- **Find options more easily.** The Tools > Options dialog provides a search box to help you find options more easily.
- **Toolbar interface improvements.** The main toolbar indicates when some toolbar items are not visible by showing drop-down lists that enable you to access the buttons that are invisible. Previously, if you had too many toolbars enabled, you could not see them all.
- **Compile a single file from a project with existing code.**
- **C/C++ Formatting style per project.** When you are choosing a project specific formatting style in project properties, you can specify the C formatting style, C++ formatting style, and the C/C++ header formatting style.

New Launchers Feature in IDE

You can create "launchers" so you can easily run your project from the project context menu with different arguments, for example, or launch it from a script. Normally when you run your application from the IDE, the executable specified in the project properties as the Run command is executed. If you create launchers, you can specify multiple commands to run, and then select them from the context menu. You can use the launchers for debugging also.

To create launchers, go to your `nbproject/private` folder and customize the Launchers File (`launchers.properties`).

In the New Launchers File dialog, select the option File Privacy if you want to store the definition of the launchers in the `nbproject/private` subfolder of the project. This option is useful if the project is shared with other developers, especially in a version control system. You can ignore the `nbproject/private` in the VCS so it is not included when you share the project. If a private launchers file exists, the launchers in the private file will override launchers of the same name in the public launcher file.

Click Finish, and a `launchers.properties` text file opens in the IDE editor. You can specify commands to run and display names to show in the IDE for running those commands. As an example, for the IDE's C/C++ sample application called Arguments, you could add the following to your `launchers.properties` file:

```
launcher1.runCommand="${OUTPUT_PATH}" "arg 1" "arg 2" "arg 3" "arg 4"
launcher1.displayName=Four Args

launcher2.runCommand=../dist/Debug/OracleSolarisStudio-Solaris-x86/arguments_1 "arg 1"
launcher2.displayName=One Arg

launcher3.runCommand=/bin/sh runMyProgram.sh
```

The file `runMyProgram.sh` might be a script that sets environment variables for example, or does anything you want.

If you want to debug your application using a launcher that runs a script, you must also specify the option `symbolFiles` for that launcher so the debugger can debug the application instead of the shell used to run the script. For the `launcher3` example above, this option could be added as follows:

```
launcher3.runCommand=/bin/sh runMyProgram.sh
launcher3.symbolFiles=${LINKER_OUTPUT}
```

When you are finished adding launcher options, save the `launcher.properties` file.

Then you can run these commands by right-clicking the project and selecting one of them. In the example above, you would right-click, select the Run As command and select the name of the command. For example, Run As > Four Args or Debug As > /bin/sh runMyProgram.sh.

IDE Code Editor Improvements

The code editor features many improvements including the following:

- “Rectangular blocks selection” on page 61
- “Clipboard History” on page 61
- “Find/Replace Enhancements” on page 61

Rectangular blocks selection

In the Editor, you can enable a mode for selecting rectangular blocks of text by pressing Ctrl +Shift+R or clicking the Toggle Rectangular Selection icon in the editor tool bar. This mode enables you to select portions of multiple lines of text without including the beginning or end of each line. For example, you can select a comment block without including the initial asterisks and copy it, or you can select and remove all the asterisks from a portion of commented out code in one action instead of deleting the asterisks line by line. You could also copy, move, or delete a column from a text table for example. If you type characters into rectangular selected text, the characters are replicated on each line replacing the selected text.

Clipboard History

You can view the last nine buffers of text that were copied to your desktop clipboard, and select one to paste. With your cursor at the point where you want to insert text, press Ctrl+Shift+D to open a popup of the clipboard entries. Use the arrow keys to navigate through the clipboard buffers and see the full contents in the window below the list of buffers. To paste the contents of a buffer, type the number of the buffer, or press Enter when the buffer is selected. Note that this buffer contains content copied from any window on your desktop, not just the IDE.

Find/Replace Enhancements

The Find/Replace feature in the editor now works completely in the Find tool bar at the bottom of the editor window, instead of a separate dialog box for replacing. The Replace field and buttons are displayed in the tool bar under the Find field and buttons. Press Ctrl+F to activate the Find tool bar and Ctrl+H to activate the Replace feature.

Code Assistance Improvements

The IDE provides many improvements in code assistance, including the following:

- “Code Assistance Cache Sharing” on page 61
- “New Project Properties Options for Code Assistance” on page 62
- “Search file system for C/C++ header files” on page 63

Code Assistance Cache Sharing

When parsing C/C++ source code, the IDE stores parse results on disk in the Code Assistance cache. When you open a project the IDE examines the cache to see if it is up to date. If the cache is up to date the IDE does not parse your project and just loads the required data for code navigation from the Code Assistance cache.

By default the Code Assistance cache resides in the `${`*userdir*`}/var/cache` folder, where `${`*userdir*`}` stands for the IDE user directory. The user directory in Oracle Solaris is in the user's `$HOME/.solstudio/ide-<`*release*`>` . The cache in the user directory cannot be shared or copied to another location.

However, if the Code Assistance cache is placed inside a project, it can be copied to another computer if that computer meets the following requirements:

- The computer's operating system is identical to the operating system where the code was parsed
- The tool collection used by the project is available in the same location on the computer

To instruct the IDE to place the Code Assistance cache inside your project metadata:

1. Add a line "`cache.location=nbproject/private/cache` " to either:
   - The project properties file (`nbproject/project.properties`)
   - The private properties file (`nbproject/private/private.properties`)

   The difference between the project properties and private properties files is that the public one (`nbproject/project.properties`) is shared in the IDE via version control system by default, while the private one (`nbproject/private/private.properties`) is not. So if you modify private properties, you will need to synchronize the private properties file with the identical file on another machine. If a project properties file is modified, a version control system can automatically synchronize it with the one on another machine for you.
2. After the properties file is modified, close and reopen the project.

   The IDE parses the project and the Code Assistance cache is placed into a private subdirectory in the project metadata.
3. Close the project and archive your nbproject/private/cache or copy it to a shared location.

   If you do not close a project before copying or zipping, some data will not be flushed to cache.

The Code Assistance cache can be copied to other projects on other machines and be used instead of waiting until the IDE parses the project. If there are some newer files on the machine the cache is being copied to, only newer files are going to be parsed.

Note - If the Code Assistance cache needs to be shared between machines that run different operating systems or different compilers, you must create a separate cache for each combination of operating system and compiler collection.

New Project Properties Options for Code Assistance

For projects created from existing sources or from a binary, the IDE now provides the following project properties to make it easier for you to use the projects in version control systems.

Transient macros	You can provide a list of macros (-D options) that are volatile -- they depend on time, date, or specific environment. These macros values will not be stored with the project's public metadata.
User Environment Variables	You can provide a list of environment variables that the project uses to pass system-specific paths. These macros environment variable values will not be stored with the project's public metadata. For projects from existing code or from binary, you may specify the list of environment variables to be used when storing project metadata. When the IDE stores the compiler options and an option value coincides with a variable value, a macros will be written instead.

Search file system for C/C++ header files

If you create a project from existing sources where the sources have not been built and does not contain any debugging information, the IDE may have trouble configuring code assistance. In this case, you can specify in the Configure Code Assistance wizard to use a special mode, Search file system for C/C++ header files. In this mode, the IDE tries to resolve failed include directives by searching the file system for headers. The wizard asks you to enter the path to search for headers. By default, the path is the project source root.

Using Breadcrumbs Navigation

Breadcrumbs enables you to keep track of your location within the IDE. The breadcrumbs navigation bar is located under the source editor window to show you nested elements relative to the cursor location. Each element is marked with an icon to identify its type. For a full list of icons and what they mean, see the Icons Used in the Classes and Navigator Windows help page in the IDE.

To use breadcrumbs, do one of the following:

- Click the arrows in the breadcrumbs bar to display a list of the nested statements or members and select one to go there in the source editor.
- Click the items in the breadcrumbs bar to go back through a history of the cursor position.
- Press Alt+Left and Alt+Right to move backward and forward through the history of the cursor position.

♦ ♦ ♦ CHAPTER 7

OpenMP API and Thread Analyzer

This chapter describes the changes for OpenMP API support and Thread Analyzer in this release of Oracle Solaris Studio.

OpenMP

This section discusses new features and updates to the OpenMP API.

OpenMP 4.0 Support

This release supports new features introduced in the OpenMP API Version 4.0, which is a major upgrade to the OpenMP API standard language specification. New OpenMP 4.0 features supported by the C, C++, and Fortran compilers in this release include the following:

- **Error Handling** - OpenMP 4.0 defines error handling capabilities to improve the resiliency and stability of OpenMP applications in the presence of runtime errors. Parallel OpenMP execution can be cleanly aborted using conditional cancellation and user-defined cancellation points.
- **Thread Affinity** - OpenMP 4.0 provides mechanisms to define where to execute OpenMP threads, resulting in better locality, less false sharing, and more memory bandwidth.
- **Tasking Extensions** - OpenMP 4.0 provides several extensions to task-based parallelism support. Tasks can be grouped to support deep task synchronization. Task-to-task synchronization is supported through the specification of task dependency.
- **Support for Fortran 2003** - The Fortran 2003 standard adds many modern computer language features. Having these features in the OpenMP specification enables users to parallelize Fortran 2003 compliant programs.
- **Sequentially Consistent Atomics** - A clause has been added to enable you to enforce sequential consistency when a specific storage location is accessed atomically.

- **User Defined Reduction** - In addition to reductions with base language operators and intrinsic procedures, OpenMP 4.0 supports user-defined reductions. Custom reductions can be defined by the programmer using the `declare reduction` directive; these reductions can then be specified in a `reduction` clause.
- **New Environment Variable** `OMP_DISPLAY_ENV` - The `OMP_DISPLAY_ENV` environment variable can be used to display the value of Internal Control Variables (ICVs) associated with the OpenMP environment variables.

Note - In this release, OpenMP 4.0 device and SIMD constructs are accepted. However, all code will be executed on the host device, and SIMD constructs might not result in the use of SIMD instructions.

For details, see the "Oracle Solaris Studio 12.4: OpenMP API User's Guide ".

For more information about the OpenMP 4.0 features, see `OpenMP Application Program Interface Version 4.0, July 2013 (http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf)` and `OpenMP 4.0.1 Examples, February 2014 (http://openmp.org/mp-documents/OpenMP_Examples_4.0.1.pdf)`.

OpenMP Related Enhancements

The following items are additional enhancements to the OpenMP API in Oracle Solaris Studio.

- **New Default Number of Threads**. -The default number of threads used to execute a parallel region has changed from two threads to the number of cores available on the machine, capped at 32.
- **Stack Overflow Detection and Diagnosis** - The existing C, C++, and Fortran compiler option `-xcheck=stkovf` has been extended to optionally enable runtime error diagnosis.

  The syntax is as follows:

  ```
  -xcheck=stkovf [:detect | :diagnose]
  ```

  If `:detect` is specified, a detected stack overflow error is handled by executing the signal handler normally associated with the error.

  If `:diagnose` is specified, a detected stack overflow error is handled by catching the associated signal and calling stack_violation(3C) to diagnose the error. If a stack overflow error is diagnosed, an error message is printed to `stderr`. This is the default behavior if nothing is specified.

  See the cc(1), CC(1), or f95(1) man page for more information about the `-xcheck=stkovf` compiler option.

Thread Analyzer

Thread Analyzer is a powerful tool that analyzes the execution of multithreaded programs and detects common threading errors, such as data races and deadlocks. With Thread Analyzer, you can debug multithreaded applications easier, leading to higher productivity. You can use Thread Analyzer with programs written using one or a combination of the following standards and frameworks:

- POSIX threads API
- Oracle Solaris threads API
- OpenMP directives

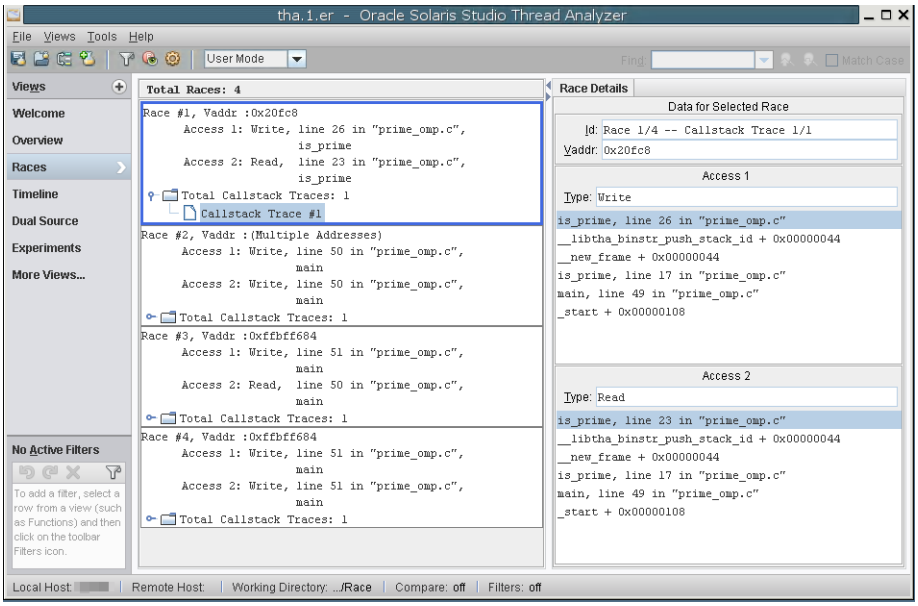
See the `tha(1)` man page and "Oracle Solaris Studio 12.4: Thread Analyzer User's Guide " for more information about Thread Analyzer.

In this release of Oracle Solaris Studio the following features have been added:

- When doing binary instrumentation for data race detection using Discover, full call stacks are displayed for data race accesses.
- Support for the `atomic_ops` API introduced in Oracle Solaris 10.
- Support for `collect -r terminate`, which forces the termination of a process when deadlock detection is on and an actual deadlock occurs.
- Thread Analyzer's user interface has been redesigned for improved data presentation and navigation along with the user interface for Performance Analyzer. For more information, see "Performance Analyzer Navigation" on page 21.

The following figure shows how Thread Analyzer displays dual call stacks when the binary has been instrumented using `discover`:

FIGURE 7-1 Thread Analyzer Window



♦ ♦ ♦ CHAPTER 8

Other Changes

This chapter describes new and changed features for other components of the Oracle Solaris Studio software.

Changes to Compilers

The following section describes changes made to the compilers and includes the following topics:

New and Changed Features Common to the Compilers

The following changes were made to the C, C++, and Fortran compilers since the previous release. Details can be found in the compiler man pages. The changes specific to the C++ compiler are detailed in Chapter 2, "C++ Compiler".

Application Performance on New Hardware

Every Oracle Solaris Studio release includes performance improvements for Oracle Sun hardware servers. This release includes expanded support for the SPARC T5, SPARC M5, SPARC M6, SPARC M10, and Intel Ivy Bridge and Haswell compiler and library performance

improvements which were first made available in the Oracle Solaris Studio 12.3 1/13 Platform-Specific Enhancements release.

- New `-xarch`, `-xchip` , and `-xtarget` values for Ivy Bridge and Haswell processors on x86.
- New `-xarch`, `-xchip` , and `-xtarget` values for SPARC T5, M5, M6, and M10+ processors.
- Support for Ivy Bridge and Haswell assembler instructions.
- Support for Ivy Bridge and Haswell intrinsic functions, which can be found in *solstudio-install-dir*`/lib/compilers/include/cc/sys</immintrin.h`.
- Default value for `-xarch=generic` set to `sse2` on x86 and x64 architectures. See "Change in Default Floating-Point Behavior for x86" on page 70 for more information.

Change in Default Floating-Point Behavior for x86

The results of floating-point computations in programs compiled using default compiler values on x86 systems might vary slightly with Oracle Solaris Studio 12.4 compared to previous releases. You might get a different result even on the same hardware and operating system. This is because the default instruction set architecture is now SSE2 for all address space models and platforms.

In Oracle Solaris Studio 12.4 and previous releases, the default address space model is `-m32` for 32-bits for Oracle Solaris. On Linux the default is `-m64` for 64-bit hardware. You can compile for the 32-bit address space model or 64 bit address space model using `-m32` and `-m64` respectively on all platforms.

The compiler optimizes code by using the `-xarch` option to determine which instructions are implemented in hardware and thus suitable for code generation. In previous Oracle Solaris Studio releases, the default was `-xarch=386` for `-m32` and `-xarch=sse2` for `-m64`.

For x86 with 32-bit addressing, if no code generation option is specified or implied with `-xarch`, `-xnative`, or `-fast`, the code generation option will be `-xarch=sse2` instead of `-xarch=386`. Thus programs using floating-point arithmetic and compiled with default `-xarch` might produce different floating-point results.

The new x86 default `-m32 -xarch=sse2` implements the same ABI as the previous default `-m32 -xarch=386`. Floating-point operands and results are passed in x87 floating-point registers. However, the following single-precision and double-precision floating-point operations are usually performed in sse2 registers.

```
+
-
*
/
sqrt
convert
```

x87 registers are still used for long double operations and hardware elementary transcendental function evaluations.

Other Compiler Changes

- Support for `-xlinkopt` on x86. Inter-module, inter-procedural code ordering optimizations for large enterprise applications tuned for modern Intel processors. An up to 5% performance boost over a fully optimized binary can be seen for large applications.
- New compiler option for x86: `-preserve_argvalues` saves copies of register-based function arguments in the stack.
- Enhanced `-xs` option to control the trade-off of executable size versus the need to retain object files in order to debug.
- Support for `-xanalyze` and `-xannotate` on Linux.
- Support for `-fopenmp` as a synonym for `-xopenmp=parallel`.
- Support for SPARC M10 values `-xchip=sparc64x`, `-xtarget=sparc64x`, `-xarch=sparcace`.
- Support for SPARC M10+ values `-xchip=sparcxplus`,`-xtarget=sparc64xplus`, and `-xarch=sparcaceplus`.
- Support for `-xarch=sparc4b` and `-xarch=sparc4c` for SPARC instruction set extensions common to SPARC M10+ processors.
- Support for Ivy Bridge values `-xchip=ivybridge`, `-xtarget=ivybridge`, and `-xarch=avx_i` on x86 platform.
- Support for Haswell values `-xchip=haswell`, `-xtarget=haswell`, and `-xarch=avx2` on x86 platform.
- New options for compilers:
  - `-g1` - Produce file and line number as well as simple parameter information that is considered crucial during post-mortem debugging.
  - `-xdebuginfo` - Controls how much debugging and observability information is emitted.
  - `-xglobalize` - Controls globalization of file static variables but not functions.
  - `-xinline_param` - Enables changing the heuristics used by the compiler for deciding when to inline a function call.
  - `-xinline_report` - Generates a report written to standard output on the inlining of functions by the compiler.
  - `-xipo_build` - Reduces compile time by avoiding optimizations during the initial pass through the compiler, optimizing only at link time.
  - `-xkeep_unref` - Keeps definitions of unreferenced functions and variables.
  - `-xpatchpadding` - Reserves an area of memory before the start of each function.
  - `-xsegment_align` - Causes the driver to include a special mapfile on the link line.
  - `-xthroughput` - Indicates that the application will be run in situations where many processes are simultaneously running on the system.

- `-xunboundsym` - Specifies whether the program contains references to dynamically bound symbols.

C Compiler

The C compiler changes include the changes that are described in "New and Changed Features Common to the Compilers" on page 69, and the following additional changes.

- Support for `-xregs=float` on x86.
- New options for the C compiler:
  - `-ansi` - Equivalent to `-std=c89`.
  - `-pedantic` - Enforces strict conformance with errors/warnings for non-ANSI constructs.
  - `-staticlib` - When used with `-library=sunperf`, links statically with the Sun performance libraries.
  - `-std` - Specifies the C language standard. `-std=c11` is the default compiler mode.
  - `-temp` - Defines the directory for temporary files.
  - `-xlang` - Overrides the default `libc` behavior as specified by the `-std` flag.
  - `-xprevise` - Produces a static analysis of the source code that can be viewed using Code Analyzer.
- Support for C11 features:
  - `_Static_assert`
  - Anonymous structs/unions
  - `_Noreturn` function assert
  - `_Thread_local` storage specifier
  - `_Alignof` operator
  - `_Alignas` alignment specifier
  - C11 specified set of characters in UCNs

See the `cc` man page and the "Oracle Solaris Studio 12.4: C User's Guide " for more information.

Fortran Compiler

The Fortran compiler supports technical and scientific application development with record-setting runtime performance and compatibility options for the Fortran77, Fortran90, and Fortran95 standards. The majority of Fortran 2003 features and OpenMP 4.0 support is

included. The Fortran compiler uses the same high-performance code generation technology as the C and C++ compilers, ensuring that the resulting application generates the highest-performance parallel code for the newest SPARC and x86-based Oracle systems.

The Fortran compiler changes include the changes that are described in "New and Changed Features Common to the Compilers" on page 69.

The following lists the new and changed features in this release of version 8.7 of the Fortran compiler. For more information, see the `f95` (1) man page and the "Oracle Solaris Studio 12.4: Fortran User's Guide ".

- The `-xM` option can be used to generate makefile dependency automatically. In conjunction with the new `-keepmod=yes` option, it allows the most optimal incremental build on Fortran application using modules. The new `-keepmod` option is used to retain a module which is not changed when compiled. The default is `-xkeepmod=yes` , which replaces the old behavior when a new module file is created each time even without any changes from the previous compilation.
- Compile time for applications using modules is substantially improved and memory overflows due to module processing are eliminated.
- `#pragma ident` can be used in a source file to identify the source version of the compiled object.
- Support for a deferred type parameter (colon) as the LEN type parameter in a character type used in a declaration. For example:

  ```
  character(LEN=:), pointer :: str
  ```

- Support for procedure pointers.
- Support for the Fortran 2003 function `C_F_PROCPOINTER()` for the `ISO_C_BINDING` module. The `C_FUNLOC()` function is extended to enable procedure pointer as an argument.
- Full support for object-oriented Fortran. Typebound procedures with the following attributes are now allowed:
  - `GENERIC`
  - `DEFERRED`
  - `NON-OVERRIDABLE`
  - `PASS`
  - `NOPASS`
- Support for the Fortran 2003 feature to enable derived type and generic function to have the same name.
- Support for the Fortran 2008 feature passing `TARGET` objects to `INTENT(IN)` pointer dummies.
- Expanded support to allow all elemental intrinsic functions (where each argument is itself an initialization expression) to be used in initialization expressions as specified in the Fortran 2003 standard. Previously, the elemental intrinsic functions used in that context were limited to those which returned type integer and character only.

- Support for `-fserialio` which specifies that the program does not perform I/O in more than one thread at a time.

Performance Library Changes

Oracle Solaris Studio Performance Library is a set of optimized, high-speed mathematical subroutines for solving linear algebra and other numerically intensive problems. Oracle Solaris Studio Performance Library is based on a collection of public domain subroutines available from Netlib at `http://www.netlib.org`. Oracle enhanced these public domain subroutines and bundled them as the Oracle Solaris Studio Performance Library.

For this release, the following changes were made:

- Tuning for performance on SPARC T5, M5, and M6 platforms and SPARC 64X+.
- LAPACK in Oracle Solaris Studio Performance Library is upgraded to version 3.4.2. All of the new features in LAPACK 3.4.2 are implemented in Oracle Solaris Studio Performance Library, including the following:
  - Extra Precise Iterative Refinement linear solvers (3.2)
  - New fast and accurate Jacobi SVD. (3.2)
  - Routines for Rectangular Full Packed format (RFP). (3.2)
  - Pivoted Cholesky. (3.2)
  - Mixed precision iterative refinement subroutines for exploiting fast single precision hardware. (3.2)
  - Computing the complete CS decomposition. (3.3)
  - Level-3 BLAS symmetric indefinite solve and symmetric indefinite inversion. (3.3)
  - `xGEQRT`: QR factorization (improved interface). (3.4)
  - `xGEQRT3`: Recursive QR factorization. (3.4)
  - `xTPQRT`: Communication-Avoiding QR sequential kernels (3.4)

  The LAPACK version in which they were introduced is indicated in parentheses.

Index