Oracle® Solaris Studio 12.4: Release Notes

Part No: E37070
May 2015

ORACLE

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support 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.

Accessibilité de la documentation

Pour plus d'informations sur l'engagement d'Oracle pour l'accessibilité à la documentation, visitez le site Web Oracle Accessibility Program, à l'adresse `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Accès au support électronique

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Using This Documentation

- **Overview** – Provides system requirements and information about known problems and other issues 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.

Feedback

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

Oracle Solaris Studio 12.4 Release Notes

This document provides information that you need to know before installing and using the product. See “Oracle Solaris Studio 12.4: Installation Guide ” for installation instructions.

System Requirements

Oracle Solaris Studio 12.4 software can be installed on Oracle Solaris 10 or Oracle Solaris 11.2 operating system on SPARC-based or x86-based platforms, or on the Oracle Linux operating system.

TABLE 1 System Requirements

System Resources	Requirements
Hardware	64-bit SPARC and x86 platforms
Operating System	Oracle Solaris 10 8/11 or Oracle Solaris 10 1/13 Oracle Solaris 11.2 Oracle Linux 5.8–5.10, 6.0–6.5 Red Hat Linux 5.8–5.10, 6.0–6.5 Oracle Unbreakable Enterprise Kernel (UEK) 2 and 3 See “Required System Software Packages” on page 10 to ensure your system has the required packages.
Distribution	SVR4 package installation for Oracle Solaris 10 IPS package installation for Oracle Solaris 11.2 RPM package installation for Oracle Linux Tar file for all platforms See “Oracle Solaris Studio 12.4: Installation Guide ” for installation instructions for all distributions.
VM Support	Oracle Solaris 11.2 Global zone running Oracle Solaris Studio on Solaris 10 or Solaris 11 as non-Global zone, LDOMs, or Kernel zones
Supported Client Systems	Client systems running IDE, Performance Analyzer, Code Analyzer, and `dbxtool` must have the minimum Java version installed

System Resources	Requirements
For installing the distributions of the IDE or Performance Analyzer for remote use	
Java Version	Java 7, minimum version 1.7.0_25 Java 8

Required System Software Packages

Before you install the Oracle Solaris Studio 12.4 release verify that the system where you are installing the software has the system packages required to run the Oracle Solaris Studio software successfully. The following table lists the required software packages for each supported operating system.

TABLE 2 System Software Required by Oracle Solaris Studio

Operating System Version	System Software Packages
Oracle Solaris 10 8/11 Oracle Solaris 10 1/13	SUNWhea SUNWarc SUNWcpcu SUNWcpp SUNWcsl SUNWlibC SUNWlibm SUNWlibms SUNWpiclh SUNWpiclr SUNWpiclu SUNWsprot SUNWlibstdcxx4 SUNWPython SUNWPython-devel
Oracle Solaris 11.2	For important information about updating pkg://solaris/consolidation/sunpro/sunpro-incorporation, see "Updating Oracle Solaris 11 System Libraries Required by Oracle Solaris Studio 12.4" in "Oracle Solaris Studio 12.4: Installation Guide ". pkg://solaris/consolidation/sunpro/sunpro-incorporation@0.5.11-0.175.2.1.0.4.0 pkg:/developer/base-developer-utilities pkg:/developer/library/lint pkg:/developer/library/xprofile pkg:/diagnostic/cpu-counters pkg:/library/glib2 pkg:/library/libxml2 pkg:/library/unixodbc pkg:/library/zlib pkg:/runtime/perl-512 pkg:/shell/bash pkg:/shell/ksh pkg:/system/header pkg:/system/library/c++-runtime pkg:/system/library/math pkg:/system/library/openmp

Operating System Version	System Software Packages
	pkg:/system/linker pkg:/system/resource-mgmt/resource-pools
Oracle Linux 5.8 - 5.10 Oracle Linux 6.0 - 6.5	Development/Libraries Package Group and 32-bit runtime support libraries including: glibc glibc.i686 glibc-devel glibc-devel.i686 elfutils-libelf-devel elfutils-libelf-devel.i686 zlib zlib.i686 libstdc++ libstdc++.i686 libgcc libgcc.i686

Installation Instructions

See the "Oracle Solaris Studio 12.4: Installation Guide " for installation instructions for all distributions.

Features That Have Been Removed in This Release

The following features have been removed in the Oracle Solaris Studio 12.4 software.

DLight Profiling Tool

DLight is a graphical profiling tool that was provided in Oracle Solaris Studio 12.3. and 12.2. You can use the Oracle Solaris Studio IDE to profile projects with some of the same tools that were part of DLight.

You can also profile your applications with Performance Analyzer which is now easier to use.

C++ Rogue Wave `tools.h++` Library

`tools.h++` is a C++ foundation class library that was provided in previous releases of Oracle Solaris Studio and Sun Studio software. The library was released in 1996 and has not been significantly updated since. The time and date classes have serious bugs that cannot be fixed.

The functionality in `tools.h++` is available with a different programming interface (API) in the C++ Standard Library provided in Oracle Solaris Studio and in the open source BOOST libraries. For information about BOOST libraries see `http://www.boost.org`.

`systent.h` Header File

This C++ header file is left over from Cfront and predates `unistd.h`. Use `unistd.h` instead.

Old Fortran Module Format

The module format generated by Forte Developer 7 Fortran 95 7.0 and earlier releases is no longer supported. The `.mod` files created with these older releases are not read by the Fortran compiler in this release.

Compiler Option `-xbinopt=prepare`

The `-xbinopt=prepare` option is obsolete. You should use `-xannotate=yes` instead.

Compiler Option `-xcrossfile`

The `-xcrossfile` option is obsolete. Use the `-xipo` option instead.

Assembler `-xarch` Options

Oracle Solaris x86 assembler `fbe` command options `-xarch=amd64` and `-xarch=generic64` are no longer supported.

You should use the `-m64` option instead.

Registration Feature

The Registration facility that enabled Sun Studio users to register with Sun Microsystems and receive late-breaking product information has been obsoleted as of 2011. The feature has been removed from the Oracle Solaris Studio product.

Features That Might Be Removed in a Future Release

The following features might be removed in a future release.

Compiler `-xarch` Options

For all Oracle Solaris Studio compilers, the command options `-xarch=amd64` and `-xarch=generic64` are obsolete and might be removed in a future release.

You should use the `-m64` option instead.

C Compiler Option `-Xs`

The `-Xs` option might be removed in a future release. If your C code requires `-Xs` to build and compile correctly, you should migrate the code to conform with at least the C99 dialect of the ISO C standard. In other words, you should be able to compile with the `-std=c99` option.

`lint` Utility `-Nlevel` Option

The `lint` utility's `-Nlevel` option for specifying the level of enhanced analysis might be removed in a future release.

`lock_lint` Utility

The `lock_lint` utility to analyze locks in multithreaded programs is obsolete and might be removed in a future release.

`libgc` Library

The `libgc` library in Studio is an obsolete garbage collection library and might be removed in a future release. You should use the Oracle Solaris `libgc` library.

F95 Interfaces with Optional Arguments in Performance Library

The optional argument feature of the `libsunperf` F95 interfaces might be removed in a future release. F95 interfaces without optional arguments continue to be supported.

VMS Compatible Library `libV77`

The `libV77` library contains two routines that conflict with the Fortran 90 intrinsic routines and do not conform to the Y2K requirement for 4-digit year. They were provided as optional routines for VMS compatibility only and might be removed in a future release.

Compiler Option `-xanalyze=code`

The `-xanalyze=code` option is deprecated and might be removed in a future release. You should compile with the `-xprevise` option instead to produce a static analysis of the source code that can be viewed using the Code Analyzer.

Static Version of `libsunmath`

The static version of the Studio math library, `libsunmath.a`, might be removed in a future release. The dynamic version, `libsunmath.so`, will continue to be available. Replace any use of the static library with the dynamic library, by replacing the options `-Bstatic -lsunmath` with `-Bdynamic -lsunmath`.

Classic C++ iostreams

Classic iostreams (`libiostream`) is the original 1986 version of iostreams, which was replaced in the 1998 C++ standard. The library is specified with the `-library=iostream` option. Classic iostreams was not a standard, so no two implementations of this library are the same and code that uses it is not portable and is not compatible with the C++ standard library. Use the iostream functions provided by the C++ Standard library.

Compiler Options for Legacy SPARC-Based Systems

Support for legacy systems might be removed in a future release. The systems include those based on the UltraSPARC I, II, IIe, III, IIIi, III+, IV, and IV+ processors.

Accordingly, the following options for the `cc`, `CC`, and `f95` compiler commands might be removed:

```
-xchip={ultra,ultra2,ultra2i,ultra2e,ultra3,
```

```
        ultra3i,ultra3cu,ultra4,ultra4plus}

-xtarget={ultra,ultra1/140,ultra1/170,ultra1/200,
          ultra2,ultra2/1170,ultra2/1200,ultra2/1300,ultra2/2170,
          ultra2/2200,ultra2/2300,ultra2e,ultra2i,ultra3,ultra3cu,
          ultra3i,ultra4,ultra4plus}
```

Compiler Option `-xdebugformat=stabs`

The `-xdebugformat=stabs` for all compilers might be removed in a future release. The only debugger format option would be `-xdebugformat=dwarf`, which is currently the default.

Runtime Checking in `dbx`

The feature Runtime Checking (RTC) in `dbx` might be removed. You can use the `discover` tool to do memory checking during runtime.

Legacy Fortran F77 Object Files

Support for object files created with the legacy F77 compiler and the `-lf77compat` linking option might be discontinued.

Legacy Array Intrinsic Runtime Libraries for Fortran

The libraries `libfmaxlai`, `libfmaxvai`, `libfminlai`, `libfminvai`, `libfprodai` and `libfsumai` on SPARC platform have not been used by the Studio Fortran compilers since the Sun Studio 10 release in 2005.

These libraries might be removed in a future release. At that point, object files and executables produced by Studio compilers earlier than the Sun Studio 10 release can no longer be used and must be recompiled with newer Studio compilers. If you have old object files and executables that require any of these libraries and recompiling is not feasible, you should keep the old compiler installation or copy the specific libraries you need from the old compiler installation into the new compiler installation.

`collect -R` Option

The `collect -R` option was formerly used to display a README file for the new features of the Performance Analyzer tools. This information is published in Chapter 3, "Performance Analysis Tools," in "What's New in Oracle Solaris Studio 12.4 ".

Third-Party Software Information

Oracle Solaris Studio 12.4 software includes third-party technologies governed by the "Oracle Solaris Studio 12.4: Third Party Notices and Licenses ".

Usage Data to Oracle

The Usage Data to Oracle feature periodically sends information on your usage of Oracle Solaris Studio components to Oracle Corporation. This information is used by Oracle Corporation to improve future Oracle Solaris Studio software releases. This information is anonymous and cannot be associated to any individual or organization.

However, if you wish to disable Usage Data to Oracle, set the `SUNW_NO_UPDATE_NOTIFY` environment variable to any value other than zero (`0`).

Documentation For This Release

The following documentation is available with this Oracle Solaris Studio 12.4 release.

- "Oracle Solaris Studio 12.4: Installation Guide ". This document includes information for installing from a tar file, a graphical package installer, and a command-line package installer.
- "What's New in Oracle Solaris Studio 12.4 ". This document describes the new features of the Oracle Solaris Studio 12.4 release.
- "Oracle Solaris Studio 12.4: Overview ". This document provides a general description about how to use the compilers and tools and how they work together.
- Online help. The online help available through the Help menu in the IDE provides task-oriented information on using all of the components of the IDE. Online help for Performance Analyzer, Thread Analyzer, Code Analyzer, and `dbxtool` is available through the Help menu in each tool.
- Man pages. These online reference manual pages describe user commands, libraries that are supplied with the compilers, and other types of commands. Man pages contain reference information, including command syntax, usage, and related commands.

You can access these documents through the `man` command in the installed Oracle Solaris Studio 12.4 software.

- Oracle Solaris Studio 12.4 manuals and tutorials. You can access these documents in HTML format or PDF format through the documentation library at http://docs.oracle.com/cd/E37069_01/

Known Problems, Limitations, and Workarounds in This Release

This section describes some of the known issues at the time of this Oracle Solaris Studio 12.4 release, and provides information about how to work around these problems.

Compiler Issues

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

Issues Common to the Compilers

This section describes known issues that apply to the `cc`, `CC`, and `f95` compilers.

Problem with OpenMP Processor Binding Using Reserved Processors

OpenMP processor binding does not respect processor reservations using the `taskset` command on Linux.

C++ Compiler Issues

This section describes known issues, problems, and workarounds for the C++ compiler in this release.

C++11 Standard Headers

The following headers are for concurrency features and are not supported in the Oracle Solaris Studio 12.4 release:

- `<atomic>`
- `<future>`
- `<thread>`
- `<mutex>`

Problems Compiling BOOST Libraries

Not all of BOOST libraries compile successfully. The results you get might depend on the version of BOOST you use, and how you exercise the libraries. BOOST libraries are included in Oracle Solaris Studio compiler testing and progress is being made toward compiling all of BOOST. If you run into a problem that prevents you from making progress, please report it.

Apache Standard Library Issue on Oracle Solaris

The Apache `stdcxx` library installed in Oracle Solaris 10 8/11 and later, and in the initial releases of Oracle Solaris 11, 11.1, and 11.2, does not work correctly with the new compiler default of `-template=no%extdef`. You will probably need to add the option `-template=extdef` to `CC` command lines that use this library. A fix for this library problem should be available in later Oracle Solaris updates. For more information, see 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`).

The Apache `stdcxx` library installed in Oracle Solaris 10 8/11 has a syntax error in header `stdcxx4/loc/_moneypunct.h`. This error was not seen by earlier compilers, but is caught by the Oracle Solaris Studio 12.4 C++ compiler. There is no way to disable the error detection.

Note - The syntax error problem was fixed in Oracle Solaris 10 1/13 and Oracle Solaris 11.1.

Ambiguity: Constructor Call or Pointer-to-Function

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

In Oracle Solaris Studio 12.2 and earlier, the compiler misinterpreted cases like the following:

```
struct S {
  S();
};
struct T {
  T( const S& );
};
```

```
T v( S() );    // ???
```

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

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

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

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

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

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

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

The first line can also be written as:

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

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

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

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

Name Mangling Linking Problems

The following conditions might cause linking problems and only apply when using the default `-compat=5` mode.

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

  Example:

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

These declarations are equivalent, but the compiler mangles the names differently. To prevent this problem, do not declare value parameters as `const`. For example, use `void foo1(int)` ; everywhere, including the body of the function definition.

- A function has two parameters with the same composite type, and just one of the parameters is declared using a `typedef`.

  Example:

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

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

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

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

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

  This section does not apply when compiling with `-compat=g` or `-std=c++11`.

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

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

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

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

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

1. You can give the variable external linkage:

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

   All instances use the same copy of `k`.
2. You can make the variable a static member of the class:

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

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

`#pragma align` Inside Namespace Requires Mangled Names

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

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

C Compiler Issues

The following issue should be noted in this release of the `cc` compiler.

`-xustr=ascii_utf16_ushort` Option is Incompatible with `-std=c11`

Specifying the flag `-xustr=ascii_utf16_ushort` results in an error if `-std=c11` (the compiler default) is in effect.

To avoid the error, you must specify an option that changes the C language dialect accepted by the compiler to an earlier version of C. The recommended options to use are `-std=c99` or `-std=c89`.

The `-xustr=ascii_utf16_ushort` option will also be accepted if you use any of the following options that change the C language dialect: `-ansi`, `-Xc`, `-Xa`, `-Xt`, or `-Xs`.

Fortran Compiler Issues

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

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

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

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

  An empty continuation line with a single ampersand is forbidden by the Fortran standard. However, with two ampersands on the same line, an empty continuation line can still be created without falling under the standard restriction. The compiler does not handle that case and gives an error. The workaround is to delete that line which only affects the readability of the program without adding any semantics. BOZ constants sometimes get truncated.
- In some relatively more complex scenarios, such as an array construct, a BOZ constant might get truncated to the default integer size of 4 bytes even though the corresponding item it is supposed to be assigned to is an 8-byte integer entity. The workaround is to use constants of correct type and kind in array construct instead of BOZ constants.
- Correction of rounding algorithm in the new release might cause differences in outputs of list-directed, name-list directed, and formatted-writes sing the `ROUND='NEAREST'` and `ROUCH='COMPATIBLE'` compared to outputs in previous releases. The rounding difference should only be in the least-significant digit.

Fortran 77 Libraries Removed

This is a reminder that the Oracle Solaris Studio 12.2 release removed the obsolete FORTRAN 77 libraries. This means that old executables compiled with the legacy Sun WorkShop `f77` compiler that depend on the shared libraries `libF77`, `libM77` and `libFposix` will not run.

Array Intrinsic Functions Use Global Registers

The array intrinsic functions `OT_PRODUCT` and `MATMUL` are highly tuned for the appropriate SPARC platform architectures. As a result, they use the global registers`%g2`, `%g3`, and `%g4` as scratch registers.

For interval arithmetic, the array intrinsics `ANY, ALL, COUNT, MAXVAL, MINVAL, SUM, PRODUCT` are also affected.

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

F95 Modules in Archive Libraries Not Included in Executable

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

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

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

`gethrtime(3F)` on Linux Platforms

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

Tools Issues

This section lists known limitations of the debugging tools and performance analysis tools.

`dbx` Limitations and Incompatibilities

`dbx` has the following limitations:

- Using older copies of `libC.so.5` or `libC.so.4` might cause problems for `dbx` in the area of C++ exceptions. Warning messages about bad stabs and unhandled exceptions might result.

  Workaround: Install the latest `libC.so.5` on all systems.
- Fortran users should compile with the `-stackvar` option to take full advantage of runtime checking.
- Some programs might not work properly with the `-stackvar` option. In such cases, try the `-C` compiler option, which enables array subscript checking without RTC.
- The `dbx` command line interpreter is an older version of the Korn shell (ksh) that does not support Code Set Independence (CSI). Multi-byte characters can be misinterpreted when typed on the `dbx` command line.
- The following features of `dbx` are not available on the Linux OS:
  - Fix and continue
  - Performance data collection on multithreaded applications.
  - Breakpoints on the following events:
    - `fault`
    - `lastrites`
    - `lwp_exit`
    - `sysin`
    - `sysout`
    - `sync`
  - Index DWARF (compiler option `-xs=no`)
- The following problems might occur when debugging programs on Linux platforms:

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

  ```
  where | head -1
  ```

  Workarounds:

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

    ```
    where
    where | head –1
    ```

  - Redirect command output to a file and then display the file contents.

    ```
    (dbx) > bag where
    (dbx) cat bag
    ```

- `dbx` does not support the following features for the GNU C and C++ compilers:
  - VL array (`gcc` 4.1 and before)
  - OpenMP
  - RTTI
  - Template definition
  - Default argument
  - `using` directive
  - `friend`
- The following issues exist for `dbx` on Oracle Linux 6:
  - Indirect reference symbols used in system libraries can sometimes cause `dbx` to set breakpoints at the reference rather than the actual function.
  - When debugging code compiled with `gcc` 4.4.4 compilers, be aware that `dbx` does not see macros defined using the `-D` compiler option.
- The stabs format for debugging information is supported in Oracle Solaris Studio, but Oracle has announced that the format might eventually be discontinued in favor of DWARF format. Oracle is not required to implement stabs support for new features or for enhancements to existing features.

  The following debugging features are not supported by the stabs format.

  - New C++11 features.
  - Parameters and local variables in optimized code.
  - Macros (when code is compiled with the `-g3` option).

  - Subsets of debugging info (generally when compiled with `-xdebuginfo={...}`).
- See "`dbx attach` Profiling (`collect -P`)" on page 28 for information about data collection problems when `dbx` is attached to a process.

Performance Analyzer and `er_print` Utility Limitations

This section describes known problems with the Performance Analyzer tool and `er_print` utility.

- The Library Visibility functionality for shared objects does not always work properly in conjunction with filtering.
- Performance Analyzer and `er_print` cannot find shared objects embedded in jar files. In addition, under some circumstances Performance Analyzer and `er_print` cannot find Java class files or source files. To work around these issues, use `addpath` to point to the directories containing the files. See the Troubleshooting section of the Help in Performance Analyzer for more detailed information.
- When you compare experiments you might see several issues:
  - Filters might not work properly.
  - Source and Disassembly views show inclusive and exclusive metrics instead of only inclusive metrics as expected.
  - Hyperlinks in Dual Source view do not always work.
  - If you set Comparison Style to Delta, the delta values are not displayed in Source and Disassembly views.
- You might see the following issues related to CPUs with variable clock rate:
  - On a system with a variable clock rate, hardware counter profiling metrics for cycle-based counters are under-reported when the processor runs at less than maximum speed and the metrics are converted to times.
  - If you compare or aggregate multiple experiments that contain hardware counter data for cycle-based counters, and the experiments were recorded on machines with different clock rates, the timing metrics shown in data views are only correct for the first experiment. However, the metrics shown in the Overview are accurate.
  - On systems with multiple CPUs running at different clock frequencies, experiments will be processed based on the clock rate for one CPU, which can lead to over-counting or under-counting on other CPUs.
- Profiling OpenMP applications that use many threads on SPARC T5, M5, and M6 systems might produce very large experiments that cannot be read. The workaround is to set the environment variable `SP_COLLECTOR_NO_OMP` before you run `collect`. Note that when you set this variable, you cannot see OpenMP constructs and metrics in the resulting experiment and the User, Expert, and Machine view modes all look the same.
- Performance Analyzer discards experiments for which no callstacks were recorded during profiling.

- Performance Analyzer cannot start if the hostname is not in `/etc/hosts`. The workaround is to ensure that the hostname is in `/etc/hosts`.

Limitations for Profiling Applications on a Remote Host

Remote login with a pass phrase is not yet supported.

`collect` Utility

This section describes known problems with the `collect` utility and data collection for the Performance Analyzer tool.

- Profiling of mixed Java and C++ applications might not be complete when using JDK 1.7.0_40 through JDK 1.7.0_59 due to a bug in the JDK. The stack for calls from Java to C++ are not properly unwound.
- Heap tracing on Linux does not trace calls to `calloc`.
- Under some circumstances, using the `~system=1` attribute for hardware counter profiling might cause the application to fail or cause the Oracle Solaris operating system to crash or reset.
- Collection of count data (`collect -c`) does not work for binaries that were compiled with `-std=C++11`.

`er_kernel` Utility

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

- `er_kernel -t` sometimes runs a few seconds longer than the requested interval.
- `er_kernel` refuses to run on an Oracle Solaris system that is running under Oracle VM to avoid an Oracle VM bug that causes a reboot.
- DTrace stack unwind sometimes omits a frame, typically when the leaf function is within its epilogue.

`dbx collector` Profiling

This section describes problems with profiling an application when using the `dbx collector` command.

- There is a limitation in using the `dbx collector` on Java. You cannot specify a Java class file for the `dbx` target. Instead, you must specify the JVM as the target and specify the class file as a parameter to the `dbx run` command. For example:

```
> dbx /path-to-your-jdk/bin/java
(dbx) collector enable
(dbx) collector java on
(dbx) run [JVM-options] [Java-class-file-to-execute]
```

- On Linux, profiling of any Java applications with `dbx` or `collect -P` might fail.
- When you perform hardware counter profiling in `dbx`, you must issue the command `collector hwprofile on` before issuing `collector hwprofile counter` or `collector hwprofile addcounter` commands that specify `hi` or `lo`. Otherwise an error will be reported.

  For example:

```
> dbx target-program
(dbx) collector hwprofile on
(dbx) collector hwprofile counter hi
```

```
> dbx target-program
(dbx) collector hwprofile on
(dbx) collector hwprofile addcounter lo
```

`dbx attach` Profiling (`collect -P`)

This section describes problems with profiling a running application when using the `dbx attach` command:

- If you attach `dbx` to a running process that was started without `LD_PRELOAD` to preload the collector library `libcollector.so` a number of errors can occur. You cannot collect any tracing data: synchronization wait tracing, heap tracing, I/O tracing, or MPI tracing. Tracing data is collected by interposing on various libraries, and if `libcollector.so` is not preloaded, the interposition cannot be done.
- If the target program installs a signal handler after `dbx` is attached to the process, and the signal handler does not pass on the SIGPROF and SIGEMT signals in Oracle Solaris or the SIGIO signal in Linux, some or all profiling data might be lost.
- If the target program uses `libcpc.so` hardware-counter experiments might fail because both the collector and the program are using the library.
- If the target program calls `setitimer` (2), clock profiling experiments can be corrupted because both the collector and the program are using the timer.
- Attaching to a target that is in the middle of a call into the `malloc` library might cause the target to fail. Requesting hardware counters when attaching greatly increases the probability of such failures.
- Attaching to a Java program fails unless the `collector java on` command or `collect -j on -P` *pid* is used.

- On Linux, attaching to a Java program with `dbx` or `collect -P` might fail.
- On Linux, attaching to a target that is in the middle of a blocking or non-blocking call might cause the target to fail.
- On Linux, attaching to a multithreaded target will not properly record data for threads that are already created at the time of the attach. No warning about missing data is provided. Note that this includes any Java targets because the JVM is multithreaded.

IDE Limitations

This section describes known limitations in the IDE.

To create a project from binary on a remote host, the remote host must have Oracle Solaris Studio 12.4 or Oracle Solaris Studio 12.3 available. This feature is not supported on earlier releases.

Code Analysis Tools Limitations

This section describes known limitations to the Code Analyzer tools.

- The `discover` and `uncover` tools do not work with an x86_64 binary compiled with `-xmodel=medium` when the program size (text + data) is large enough to extend beyond the maximum range of `-xmodel=small` (2^{32} - 2^{24} - 1).
- The `discover` tool doesn't work on an ancillary file, which is a file containing debug information created with the linker option `-z ancillary` in Oracle Solaris 11.2.
- The `discover` tool does not report UMRs and PIRs with compiled with C++11.
- To use any of the Code Analysis tools on Oracle Linux, the original binary must be compiled and linked with the `-xannotate=[yes]` option.