Oracle® Developer Studio 12.6: GCC Compatibility Guide

June 2017

Part No: E77792

This document discusses the GCC (Gnu Compiler Collection) compatibility with Oracle Developer Studio compilers and tools.

This guide is intended for people who would like to take source code that normally builds with the GNU C and C++ compilers and build it with the Oracle Developer Studio C and C++ compilers. This strategy might include using Oracle Developer Studio and GNU compilers for different parts of the same application. This strategy is also useful for people who need to maintain source code in such a way that it is compatible between both kinds of compilers.

In this document, the term GCC is used to refer to the Gnu Compiler Collection, which includes (among others) a C and a C++ compiler. The term `gcc` refers to the GNU C compiler and `g++` refers to the GNU C++ compiler.

Note - Although many implementation details are discussed in this document, the definitive source for supported features are the *Oracle Developer Studio 12.6: C User's Guide* and the *Oracle Developer Studio 12.6: C++ User's Guide*.

General Compatibility Concepts

This section discusses basic concepts affecting the compatibility between Oracle Developer Studio and GCC compilers.

Platforms and ABIs

The C or the C++ language standards do not define any formal ABI (Application Binary Interface). However, within a specific platform and pointer size, a de facto ABI enables compatibility between modules written with C-level exported interfaces.

In this context, a platform is a combination of two items:

- Operating system (for example, Oracle Linux or Oracle Solaris).
- Chip family (SPARC or x86).

The pointer size refers to whether the binaries are built with the 32-bit ABI or the 64-bit ABI. Some platforms might support only one pointer size, but currently most platforms supported by the Oracle Developer Studio product support both 32-bit and 64-bit programs.

Oracle Developer Studio and `gcc` support the `-m32` and `-m64` options for selecting the 32-bit and 64-bit ABIs respectively. On Solaris 10 and Oracle Solaris 11, the default mode is 32-bit. On Oracle Linux, the default mode is 64-bit.

To implement many features of the C++ language, the compiler must generate ELF symbols and other binary data that are specific to the compiler implementation and not covered by a multi-vendor platform-wide ABI document.

Some features of the compiler that are selected by compile-time or link-time options can result in additional external symbol references or other changes in the binary output of the compiler that is beyond the usual C level interfaces defined in the ABIs. For these features, you might have to link your program with the same compiler used to create the object files. Some features might also prevent linking with object files from other compilers.

Compatibility Summary

You can generally compile C source files with Oracle Developer Studio or `gcc` and freely mix their object files to link either an executable or a shared library. This section describes some exceptions.

For C++, you can choose a `g++` compatible mode of the Oracle Developer Studio C++ compiler and mix shared libraries and executables built with different compilers, but you cannot mix object files from different compilers. Details are described below.

The Oracle Developer Studio 12.6 release has a specific issue with `g++` compatibility between the 4.*x* library and the 5.*x* library ABIs. For more information, see "GNU ABI Compatibility" on page 19.

ABI References

The binary code generated by the Oracle Developer Studio compilers is described in a variety of documents:

- SPARC ABI: `http://sparc.org/technical-documents`
- SPARC Assembly Language Reference Manual : Chapter 6. Writing Functions `http://docs.oracle.com/cd/E53394_01/html/E54833/index.html`
- x86 ABIs: `https://github.com/hjl-tools/x86-psABI/wiki/X86-psABI`
- Oracle Solaris 64-bit Developer's Guide
  - AMD64 ABI Features: `https://docs.oracle.com/cd/E53394_01/html/E61689/fcowb.html`
  - SPARC V9 ABI Features: `https://docs.oracle.com/cd/E53394_01/html/E61689/advanced-2.html`

C++ GNU ABI Mode and Sun ABI Mode

The Oracle Developer Studio C++ compiler has two modes of operation. The older Sun mode has backward compatibility with binaries that are built with older compilers and is not compatible with g++. The Sun mode is enabled with `-compat=5`. Newer modes of operation are enabled for C++ 11 compliance using the `-std` option. For more information about these options, see the *Oracle Developer Studio 12.6: C++ User's Guide*.

Standards Conformance Overview

The Oracle Developer Studio and GCC compilers can compile in specific modes that conform to different language or operating system standards. These include C and C++ language standards and various POSIX, UNIX, SUS, or XPG standards. These modes have the following effects:

- The compiler enables or disables specific syntax that exhibits different behaviors for different language standards.
- The compiler enables or disables specific keywords and variant spellings of keywords. For example, `__restrict` and `restrict`.
- In response to CPP symbols controlled by the compiler, the system headers can either hide or make visible the specific APIs that are a part of the selected language standards or the Operating System standards.

Configure Scripts

Open source projects often use the `autoconf` package to create a script called `configure` that is used to detect platform-specific properties and set up correct macros and Makefiles to match the platform you are building on.

The Oracle Solaris system header files are designed to exclude Oracle Solaris extensions if the application requests any POSIX compilation modes using macros like `_POSIX_SOURCE` or `__XPG7`. You can perform the following task in order to request extensions to be added back to the system headers.

Add `AC_USE_SYSTEM_EXTENSIONS` flag to your `configure.ac` file, and rerun `autoconf`. This generates a `configure` script which will add the `__EXTENSIONS__` macro when the configure script is run on Oracle Solaris. The system headers then enable extensions even in the conforming mode. Linux has a similar macro called `_GNU_SOURCE`.

For more information about the flag, see autoconf documentation.

Assembler Compatibility

The SPARC and x86 assemblers act very differently in terms of machine code, and in terms of pseudo-ops. Therefore, the compatibility issues between the GNU assembler (`gas`) and the Oracle Developer Studio and Oracle Solaris assemblers will be different on different platforms.

The same source code is used to produce the assemblers that ship with Oracle Solaris (SPARC and x86) and the assemblers that ship with Oracle Developer Studio (SPARC and x86). So, you should expect similar compatibility issues when dealing with either the Oracle Developer Studio or Oracle Solaris assemblers.

x86 Assembler

Note the following x86 assembler issues when switching between the Oracle Developer Studio and Oracle Solaris assemblers and the GCC assembler.

- `gcc` can often infer opcode suffixes, whereas Oracle Developer Studio insists they be explicitly provided. Being more explicit satisfies both. For example, change `mov` to `movw` and `shr` to `shrw`.
- '`#`' introduces comments in `gcc` assembler files, but has historically been expected to introduce preprocessing directives in Oracle Developer Studio assembler files.

For more information, see x86 Assembly Language Reference Manual

SPARC Assembler

The official SPARC assembly format is defined by The SPARC Assembly Reference Language Manual.

Assembler Directives Related to ELF Sections

The `.section` directive takes different arguments in the SPARC assembler. The attribute flags are defined using explicit tokens instead of a character string as in the GNU assembler (`gas`). For example, with `gas`, a `.section` directive would appear as follows:

```
.section .init,"aw"
```

When using the Oracle Developer Studio or Oracle Solaris SPARC assembler, the directive would appear as follows:

```
.section ".init",#alloc,#write
```

Note - The SPARC assembler supports the `.pushsection` and `.popsection` directives, but not `.previous` directive.

Pseudo-Op Issues

The `.symver` pesudo-op is supported in the SPARC assembler for GNU compatibility.

The `.uleb128` and `.sleb128` pseudo-ops are supported.

SPARC Assembler Resources

For more information, see the following resources:

- Descriptions of the current SPARC instruction sets: http://www.oracle.com/technetwork/server-storage/sun-sparc-enterprise/documentation/sparc-processor-2516655.html
- SPARC ABI Documentation: http://sparc.org/technical-documents/

Header File Compatibility

GCC and Oracle Developer Studio compilers predefine different symbols.

Use the following command to view the symbols that are predefined by `gcc` for C and C++:

```
$ gcc -E -dM -xc /dev/null
$ g++ -E -dM -xc++ /dev/null
```

Use the following command to view the symbols that are predefined by Oracle Developer Studio C and C++ compilers:

```
$ cc -xdumpmacros -E /dev/null
$ CC -xdumpmacros -E /dev/null
```

Use of various compiler options can affect the predefined macros and their values for all compilers. The `-m32`|`-m64` options and the language options (`std=`*v*) in particular affect predefined macros.

This output also includes source code defining the `_GNU_SOURCE` macro, which is used to enable header file declarations for various non-portable functions, defined mostly in `glibc`. Fore more information, see http://stackoverflow.com/questions/5582211/what-does-define-gnu-sourceimply.

Preprocessor Compatibility

The Oracle Developer Studio C and C++ compilers have their own built-in implementations of the C preprocessor that are used by default. These preprocessors have the following extensions which are compatible with `gcc`.

The Oracle Developer Studio preprocessors:

- `#warning`

- `#include_next` (to implement wrapper headers)

Some code that depends on traditional mode behavior of the preprocessor will also encounter differences between the Oracle Developer Studio and GCC compilers. The traditional mode in `gcc` is described at https://gcc.gnu.org/onlinedocs/cpp/Traditional-Mode.html.

Although you can achieve traditional behavior using `/usr/lib/cpp` on Oracle Solaris if you require that behavior, the best practice is to replace those dependencies with more modern usages. `/usr/lib/cpp` is not intended to be a fully `gcc`-compatible preprocessor.

The following examples show the output that are sometimes seen in source code.

EXAMPLE 1 Token-Pasting Using Empty Comments

```
FOO/**/BAR
```

The expected result is “FOOBAR” in the output. This does not work in either `gcc` or Oracle Developer Studio unless you take special steps to enable traditional mode. In Oracle Developer Studio C, this means specifying the `-Xs` option. In `gcc`, you must specify the `-traditional-cpp` option. It is better to update the code to use the standard `##` token-pasting operator defined in C and C++.

EXAMPLE 2 Inter-Token Spacing

```
#define FOO foo
#define BAR bar
FOO-BAR
```

The expected output here is “FOO-BAR”, not “FOO - BAR”. The preprocessor might or might not add extra spaces around the '-' symbol. The `gcc` compiler does not add spaces, and some code depends on that behavior when using the compiler as a preprocessor with the `-E` or `-P` options. To get the traditional behavior, you can use `-Xs` with the Oracle Developer Studio C compiler (not supported in Oracle Developer Studio C++) or you can use `/usr/lib/cpp` directly.

Compiler Compatibility

This section discusses compiler features and other behaviors that affect compatibility between Oracle Developer Studio and GCC.

Implementation Defined Behavior

Some parts of the C and C++ standards are left as “implementation defined behavior”. These details are defined in the GCC and Oracle Developer Studio documentation, and they differ in some ways.

Both bit-fields and enumerated types can be either signed or unsigned by the choice of the compiler. For `enums`, this choice can vary based on the list of enumerated values. The behavior of Oracle Developer Studio C and C++ is different from the behavior of `gcc`.

Signed and Unsigned `int` Bit-fields

Bit-fields which are declared as `int` (not `signed int` or `unsigned int`) can be implemented by the compiler using either signed or unsigned types. This makes a difference when extracting a value and deciding whether to sign extend it.

The Oracle Developer Studio compiler uses unsigned types for `int` bit-fields and the `gcc` compiler uses signed types. Use the `gcc -funsigned-bitfields` flag to control this behavior.

For more information, see the sixth list item at https://gcc.gnu.org/onlinedocs/gcc-3.3.6/gcc/Non_002dbugs.html.

Signed and Unsigned `enum` Types

The `gcc` compiler performs a kind of optimization related to the types of enums. If an enum type has no negative values defined for it, then the internal type used by the compiler will be an unsigned type. If you cast a negative value into that type, it will be treated as a large unsigned value.

The C language standard enables the compiler to choose either signed or unsigned `int` types to store enums, as long as the type is compatible with all the defined enum values. The Oracle Developer Studio C compiler always uses signed `int` types for enums.

Note - The type of an enumeration value in standard C is defined to be `signed int`.

In Oracle Developer Studio 12.6, the C compiler supports the option `-features=gcc_enums` to implement the same behavior as `gcc` for enums.

Note that this behavior is quite different from the way enums are implemented when the `-fshort-enums` flag is in effect. Oracle Developer Studio 12.6 has the `-fshort-enums` option similar to `gcc`. However, it is not recommended. It also affects the layout of structs in headers for libraries that might not have been compiled with `fshort-enums`. The result would be buggy code.

Characters in Identifiers

In Oracle Developer Studio 12.6, C++ compiler added the ability to use unicode character in symbols with `\u` and `\U` prefixes.

Valid Keywords

Oracle Developer Studio and GCC differ in the way keywords have spellings. For example, the C 99 keyword `restrict` can be spelt as `restrict`, `_restrict` or `__restrict__`. The spellings that are valid at any time will be affected by the language mode, such as C 99 vs C 11. If you have an application that uses a keyword with a specific spelling, then you can use a command line argument to define it as a macro. For example, `-D__restrict=restrict`.

GCC defines a special category of hidden keywords that have leading and trailing underscores. These variant spellings are enabled in all compilation modes so that system headers can use them regardless of whether the current standards mode allow the main keyword or not. For example, `__restrict__` will be enabled in all modes, even when `restrict` is not valid.

Compiler Standards Conformance

The following compiler options affect the standards conformance modes:

- `-pedantic` (and `-Wpedantic`, `-pedantic` errors).
- `-std`. For example, `-std=c11` or `-std=c++11` or `-std=gnu11`.
- `-D__EXTENSIONS__`, that enables Oracle Solaris extension APIs even in conformance modes.

Sometimes, the Oracle Solaris system headers react in different ways to GNU and Oracle Developer Studio compilers based on the different CPP symbols that are predefined.

The Oracle Developer Studio 12.6 C++ compiler defines the `__STDC__` macro to the value 0 on the Oracle Solaris platform. Other C++ compilers (including g++) set this macro to 1. If your code depends on this value, it can be overidden on the command line with `-D__STDC__=1`.

Strict and Feature-Enabled `-std` Options

The Oracle Developer Studio 12.6 C compiler release includes new language standard options that are compatible with GCC. GCC defines different variations of language standards to differentiate between *strict conformance* and *extensions enabled*. When compiling in strict mode, `gcc` defines a `gcc`-specific symbol called `__STRICT_ANSI__`. The latest Oracle Developer Studio release defines this symbol in the same language modes where `gcc` defines it.

For more information, see Options Controlling C Dialect.

TABLE 1 Types of `-std` Options

`-std` Option Family	`gcc`	Oracle Developer Studio 12.5	Oracle Developer Studio 12.6
Strict Conformance `-std=c99` (or `c11`, etc)	Implemented `__STRICT_ANSI__`	Implemented `No __STRICT_ANSI__` (default)	Implemented `__STRICT_ANSI__`
Feature-Enabled `-std=gnu99` (or `gnu11`, etc)	Implemented (default)	Not Implemented	Implemented (default)

As the table shows, the default value of the `-std` option moves to the *with extensions* variant instead of the *strict* variant.

On Linux, the `gcc` symbol `__STRICT_ANSI__` was already defined by Oracle Developer Studio, but since the default `-std` option is changing to `non-strict`, that behavior will change in the new release. In order to retain the old behavior on Linux, you can specify `-std=c11` option.

In Oracle Developer Studio 12.6, the only difference between these modes is the presence of `__STRICT_ANSI__`. Extensions might be disabled as necessary in order for the compiler to implement a strict standards conformance mode.

The following features provide compatible experience for most users.

- Implementation of gnu language modes.
- `__STRICT_ANSI__` defined in non-gnu modes.
- Transition of the default mode from `strict` to `gnu`.

The following features might provide incompatible experience to the users.

- Users on Linux who depend on `__STRICT_ANSI__`.
- Users who explicitly specify the `-std` option will now have `__STRICT_ANSI__` set.

Inlining

The Oracle Developer Studio C compiler and GCC C compiler had support for declaring functions as inline before the behavior was standardized. Both these compilers implement the new standard behavior and have flags to enable backward compatibility to the inline behavior.

GCC and Oracle Developer Studio have the options `-fgnu89-inline` and `-xfeatures=extinl` respectively. For more information, see Inline Functions in the GCC documentation.

C Language Extensions

Many C language extensions in Oracle Developer Studio are also implemented in Oracle Developer Studio C++. Some of the C extensions are documented in "Extensions" in *Oracle Developer Studio 12.6: C User's Guide*. For the list of `gcc` language extensions, see Extensions to the C Language Family in the GCC documentation.

Some of the items listed in the `gcc` table are now standard features of a relevant language standard and are listed in Table 2, "GCC Extensions implemented in Oracle Developer Studio," on page 9 as C 11, C 99, C++ 03, C++ 11, and C++ 14.

Features of a newer language standard are often available as an extension when compiling according to older language standards unless either they conflict with existing code or you are using an option to enable a strict conformance mode. For more information about features that are available in different modes, see the documentation for that compiler.

The following table lists the extensions that are implemented in Oracle Developer Studio.

TABLE 2 GCC Extensions implemented in Oracle Developer Studio

GCC Extension	Oracle Developer Studio Implementation Status
Statement Expressions: Putting statements and declarations inside expressions.	Implemented.
Local Labels: Labels local to a block.	Implemented in C only.
Labels as Values: Getting pointers to labels, and computed `gotos`.	Implemented in C only.
Nested Functions: As in Algol and Pascal, lexical scoping of functions.	Not Implemented.
Constructing Calls: Dispatching a call to another function.	Not Implemented.
Typeof: referring to the type of an expression.	Implemented. New in Oracle Developer Studio 12.6 C++ C++ 11 defines `decltype` for this. In Sun (`-compat=5`) mode, C++ omits the `typeof` keyword, but still supports `__typeof` and `__typeof__`.
Conditionals: Omitting the middle operand of a '?:' expression.	Implemented in C only.
`__int128`: 128-bit integers—`__int128`.	Not implemented.
Long Long: Double-word integers—`long long int`.	Implemented in C and C++.
Complex: Data types for complex numbers.	untyped `_Complex` defaults to `double` for compatibility with `gcc`. Implemented in C only.
Floating Types: Additional Floating Types.	Oracle Developer Studio C and C++ implement the 128-bit `long double` type, but not with the type named `__float128`.
Half-Precision: Half-Precision Floating Point.	Not implemented.
Decimal Float: Decimal Floating Types.	Not implemented.
Hex Floats: Hexadecimal floating-point constants.	C 99. Implemented in C only.
Fixed-Point: Fixed-Point Types.	Not implemented.
Named Address Spaces: Named address spaces.	Not implemented.

GCC Extension	Oracle Developer Studio Implementation Status
Zero Length: Zero-length arrays (general zero length arrays).	`int foo[0];` Implemented in C++ in GNU compatibility mode, or with `-features=zla`. Implemented in Oracle Developer Studio 12.6 C compiler with `-features=zla`.
Zero Length: Zero-length arrays (flexible array members).	`int foo[]; //` at the end of a struct Implemented in C++ in GNU compatibility mode, or with `-features=zla`. Implemented in C.
Empty Structures: Structures with no members.	Implemented in Oracle Developer Studio 12.6 for C compiler. Implemented. For C, requires `-features=extensions`.
Variable Length: Arrays whose length is computed at run time.	C 99. Implemented in C and C++.
Variadic Macros: Macros with a variable number of arguments	C 99. Implemented in C and C++. Oracle Developer Studio implements the `gcc` extension for supplying a user-defined name for the variable macro arguments. `gcc` extensions for missing variables arguments are not implemented in C++.
Escaped Newlines: Slightly looser rules for escaped newlines.	Implemented in Oracle Developer Studio 12.6 for C compiler. Not implemented.
Subscripting: Any array can be subscripted, even if not an `lvalue`.	C 99. Standard C++. Implemented in C and C++.
Pointer Arith: Arithmetic on void-pointers and function pointers.	Implemented in C only. Warning is generated.
Pointers to Arrays: Pointers to arrays with qualifiers work as expected.	Not implemented.
Initializers: Non-constant initializers.	C 99. Standard C++. Implemented.
Compound Literals: Compound literals give structures, unions, or arrays as values.	C 99. Implemented in C.
Designated Inits: Labeling elements of initializers.	C 99. Implemented in C.
Case Ranges: `` `case 1 ... 9` `` and such.	Implemented.
Cast to Union: Casting to the `union` type from any member of the union.	Not implemented.
Mixed Declarations: Mixing declarations and code.	C 99. Standard C++. Implemented.
Attribute Extensions	`__has_attribute()` can be used to test for recognized attributes. For more information, see "Attributes" on page 13.
Function Prototypes: Prototype declarations and old-style definitions.	Implemented in C only. Not relevant to C++.
C++ Comments: are recognized.	Implemented in C.
Dollar Signs: Dollar sign is allowed in identifiers.	Implemented. Requires `-features=iddollar`.
Character Escapes: '\e' stands for the character <ESC>.	Not implemented.
Alignment: Inquiring about the alignment of a type or variable.	C 11, spelled `_Alignof`. `__alignof__` supported in C and C ++. C++ supports `alignof` in C++ 11 mode.
Inline: Defining inline functions (as fast as macros).	C 99 and Standard C++. Implemented. Before GCC implemented the standard, it created a static function body instead of an external one. If your code depends on this,

GCC Extension	Oracle Developer Studio Implementation Status
	you can use the C option `-features=no%extinl` to get similar behavior from the Oracle Developer Studio C compiler.
Volatiles: What constitutes an access to a volatile object.	Implemented. Compatible with GCC.
Using Assembly Language with C: Instructions and extensions for interfacing C with assembler.	Implemented. C and C++ implement `gcc`-compatible `asm()` statements, including constraints, `asm()` labels, and explicit register variables.
Alternate Keywords: `__const__`, `__asm__`, among others, for header files.	Implemented. Oracle Developer Studio 12.5 C++ added `__asm` and `__volatile` keyword spellings.
Incomplete Enums: `enum foo;`, with a subsequent definition.	C++ 11. Implemented in C and C++. Must be C++ 11 mode.
Function Names: Printable strings which are the name of the current function.	Oracle Developer Studio compilers support `__func____FUNCTION__` and `__PRETTY_FUNCTION__`.
Return Address: Getting the return or frame address of a function.	Not implemented.
Vector Extensions: Using vector instructions through built-in functions.	See "SIMD Vector Support" on page 12.
Offsetof: Special syntax for implementing `offsetof`.	Implemented in Oracle Developer Studio 12.6 for C and C++ compilers
`__sync` Builtins: Legacy built-in functions for atomic memory access.	Implemented. Use `-xatomic=studio`. See "Atomics" on page 21. (`gcc` deprecates these functions in favor of the standard `__atomic` builtins.)
`__atomic` Builtins: Atomic built-in functions with memory model.	Implemented. New in Oracle Developer Studio 12.6. Use `-xatomic=studio`. See "Atomics" on page 21.
Integer Overflow Builtins: Built-in functions to perform arithmetic and arithmetic overflow checking.	Not implemented.
x86 Specific Memory Model Extensions for Transactional Memory: x86 memory models.	Not implemented.
Object Size Checking: Built-in functions for limited buffer overflow checking.	Not implemented.
Pointer Bounds Checker Built-ins: Built-in functions for Pointer Bounds Checker.	Not implemented.
Cilk Plus Built-ins: Built-in functions for the Cilk Plus language extension.	Not implemented.
Other Builtins: Other built-in functions.	`__builtin_constant_p()` can test whether something is a compile-time constant. Most other built-ins are not yet implemented by Oracle Developer Studio. `__builtin_unreachable()` was added as a no-op to C++. It might be added to C. `__builtin_expect()` was added in Oracle Developer Studio 12.6 in C compiler.
Target Builtins: Built-in functions specific to particular targets.	Not implemented.
Target Format Checks: Format checks specific to particular targets.	Not implemented.
Pragmas: Pragmas accepted by `gcc`.	`#pragma once` implemented. Others not implemented.
Unnamed Fields: Unnamed struct/union fields within structs/unions.	C++ 11. Implemented in C and C++.
Thread-Local: Per-thread variables.	C 99. C++ 11. Implemented in C and C++.
Binary Constants: Binary constants using the '`0b`' prefix.	Implemented in C++ only.

SIMD Vector Support

Oracle Developer Studio supports some architecture-specific intrinsics to encode vector operations using CPU-specific instructions. The data types for these are created using the `vector_size` attribute or by including an appropriate header file as described in “Header File Compatibility” on page 5. Some C operators are supported on such types, but in some cases you might have to use the CPU-specific intrinsics.

For a discussion of types like `_m128` and operations on them, see the SPARC64 X section in “SIMD Intrinsics” in *Oracle Developer Studio 12.6: C User’s Guide*.

Oracle Developer Studio implements a set of x86 vector intrinsics with names that begin with the prefix `_mm_`. On Linux they can be accessed using `xmmintrin.h` which is bundled with `gcc`. On Oracle Solaris, they can be accessed by including `sys/mmintrin.h` which is bundled with the Oracle Developer Studio compilers. For more information, see “Compiler Support for Intel MMX and Extended x86 Platform Intrinsics” in *Oracle Developer Studio 12.6: C User’s Guide*

C++ Specific Features

The Oracle Developer Studio C++ compiler supports the `-features=cplusplus_redef` option that enables a non-standard value of the `__cplusplus` macro for source codes that requires that setting. For more information, see “–xrestrict[=f]” in *Oracle Developer Studio 12.6: C++ User’s Guide*.

Some other GNU extensions for C++ are:

- `long long` constants in enumerations.
- Allowing a `typedef` for `void` as a function parameter.

```
typedef void VOID;
int foo(VOID);
```

- `_Bool`, which is equivalent to `bool` on Oracle Linux when `<stdbool.h>` is included.
- `extern` template.
- `long long` bit-field.
- `pragma` pack push/pop.

For information about C++ Extensions, see Extensions to the C++ Language in the GCC documentation. Information about the same functionality in the Oracle Developer Studio compiler can be found in the following table.

TABLE 3 GNU C++ Extensions

GNU C++ Extension	Oracle Developer Studio Implementation Status
C++ Volatiles: Determine what constitutes an access to a volatile object.	Compatible Implementation.
Restricted Pointers: C 99 restricted pointers and references.	Implemented.
Vague Linkage: Where G++ puts inlines, vtables and the like.	Oracle Developer Studio also uses COMDAT for these, probably in a slightly different way.
C++ Interface: You can use a single C++ header file for both declarations and definitions.	Not implemented. Deprecated in `gcc`.
Template Instantiation: Methods for ensuring that exactly one copy of each needed template instantiation is emitted.	Oracle Developer Studio also uses COMDAT for these, probably in a slightly different way.
Bound member functions: You can extract a function pointer to the method denoted by a ‘->*’ or ‘.*’ expression.	Not Implemented.

GNU C++ Extension	Oracle Developer Studio Implementation Status
C++ Attributes: Variable, function, and type attributes for C++ only.	See “Attributes” on page 13.
Function Multiversioning: Declaring multiple function versions.	Not Implemented.
Namespace Association: Strong using directives for namespace association.	Not implemented. Deprecated in `gcc` in favor of a standard C++ 11 feature.
Type Traits: Compiler support for type traits.	Not Implemented.
C++ Concepts: Improved support for generic programming.	Not Implemented.
Java Exceptions: Tweaking exception handling to work with Java.	Not Implemented.
Deprecated Features: Things that are scheduled to be removed from C++.	Not Implemented.
Backwards Compatibility: Compatibilities with earlier definitions of C++.	Not Implemented.

Attributes

The full documentation for attributes implemented in the Oracle Developer Studio 12.6 compilers is in “Supported Attributes” in *Oracle Developer Studio 12.6: C User’s Guide* and “Supported Attributes” in *Oracle Developer Studio 12.6: C++ User’s Guide*.

The `__has_attribute()` built-in can be used with both `gcc` and Oracle Developer Studio compilers to test for recognized attributes.

The following attributes are new in Oracle Developer Studio 12.6 C++: `aligned`, `deprecated`, `weak`(`alias`), `weakref`, `packed`, `tls_model`, `vector_size`, and `visibility`.

The Oracle Developer Studio compilers do not support all of the syntaxes for attributes.

TABLE 4 GCC Attributes

GCC Attribute Category	Attributes supported in Oracle Developer Studio
Function Attributes: Declaring that functions have no side effects, or that they can never return.	`alias`, `aligned`, `always_inline`, `const`, `constructor` / `destructor`, `deprecated` (C++ 14), `malloc`, `noinline`, `nonnull` (C++), `noreturn`, `nothrow` (c++ only), `pure`, `regparm/ssregparm` (C++ only), `returns_twice`, `transparent_union`(C++ only), `visibility`, `warn_unused_result` (C++), `weak` (and `alias`), `weakref` (C ++)
Variable Attributes: Specifying attributes of variables.	`aligned`, `common` (C++), `deprecated`, `mode` (C++), `nocommon` (C++), `packed` , `section` (C), `tls_model`, `vector_size`, `weak`
Type Attributes: Specifying attributes of types.	`aligned`, `deprecated`, `packed`, `visibility`
Enumerator Attributes: Specifying attributes on enumerators.	`deprecated`

Command-Line Options

Both the Oracle Developer Studio and the `gcc` compilers implement traditional compiler options like `-g`, `-c`, `-o`, and many others. The following table describes the options in the Oracle Developer Studio compilers that have been implemented specifically to be compatible with `gcc`.

The following options in Oracle Developer Studio are compatible with `gcc`.

TABLE 5 Compatible Options

Option	Description
`-std`	The `-std` option selects a language standard like C 11, C++ 11, and others.
`-pedantic`	Issues errors or warnings for technical violations of the standard that would otherwise be accepted. This option is new in the Oracle Developer Studio 12.6 C++ compiler.
`-m32` / `-m64`	Selects 32-bit or 64-bit binary output.
`-shared`	Produce a shared library. In Oracle Solaris Studio 12.4, the `-shared` option acted like an alias for `-G`. In Oracle Developer Studio 12.6, this option acts like the `-shared` option in `gcc`. See Table 6, “Options with Differences,” on page 14.
`-gz=[=type]`	Produce compressed debug sections in DWARF format, if that is supported. If `type` is not given, the default type depends on the capabilities of the assembler and linker used. `type` might be one of `none` (do not compress debug sections), `zlib` (use zlib compression in ELF gABI format), or `zlib-gnu` (use `zlib` compression in traditional GNU format). If the linker does not support writing compressed debug sections, the option is rejected. Otherwise, if the assembler does not support them, `-gz` is silently ignored when producing object files.
`-fsemantic-interposition`	Controls whether exported symbols need to be replaced using dynamic interposition at runtime. For example, if the symbols need to be replaceable, then the symbols must not be inlined by the optimizer.
`-fshort-enums`	Use smaller types for enums. This also results in automatically packing enums in structs.

The following options work differently between Oracle Developer Studio and GCC.

TABLE 6 Options with Differences

Oracle Developer Studio Option	GCC Option	Description
`-xM`	`-M`	Print make-style dependencies for a source file. In `gcc`, `-xM` performs another function. In Oracle Developer Studio, `-M` selects a linker map file.
`-B`(static| dynamic)	`-Wl,-B`(static| dynamic)	GNU Linker supports this behavior, but `gcc` does not. Use `-Wl` to pass the option to GNU `ld`. In `gcc`, `-B` performs another function.
`-G`	`-shared`	When producing a shared library, `-shared` in `gcc` adds C++ library dependencies. The `-G` option in the Oracle Developer Studio compilers does not add them.

The following `gcc`-style options are automatically translated into corresponding options in Oracle Developer Studio. This list of options can be seen by passing this option to the following option to the C or C++ compiler: `-xhelp=gccflags`.

TABLE 7 Automatically Translated Options

`gcc` option in Oracle Developer Studio	Behavior in Oracle Developer Studio compilers
`-MM`	Same as `-xM1`
`-Ofast`	Same as `-fast`
`-Wall`	Same as +w2 (C++ only)
`-Wall`	Same as `-v` (C only)
`-Werror`	Same as `-errwarn=%all`
`-Wpedantic`	Same as `-pedantic`
`-fno-elimininate-unused-debug-types`	Accept and ignore

`gcc` option in Oracle Developer Studio	Behavior in Oracle Developer Studio compilers
`-fopenmp`	Same as `-xopenmp`
`-fPIC`	Same as `-KPIC`
`-fpic`	Same as `-Kpic`
`-fplan9-extensions`	Accept and ignore (C only).
`-fplugin-arg-name=`*t*	Warn and ignore (C only)
`-fplugin=`*t*	Warn and ignore (C only)
`-fsigned-char`	Same as `-xchar=signed` (C only)
`-fsyntax-only`	Same as `-xe`
`-funsigned-char`	Same as `-xchar=unsigned` (C only)
`-gdwarf-version`	Same as `-xdebugformat=dwarf`
`-gz[=cmp-type]`	Same as `-xcompress=debug` with `-xcompress_format=cmp-type`. `-gz` with no sub-option is equivalent to `-gz=zlib`.
`-gstabs`	Same as `-xdebugformat=stabs`
`-gstabs+`	Suggest using `-xdebugformat=stabs`; option ignored
`-iplugindir=`*t*	Warn and ignore
`-march=`*a*	Same as `-xtarget=`*a*
`-mcpu=`*a*	Same as `-xtarget=`*a*
`-mfmaf`	Same as `-fma=fused`
`-mno-vis`	Same as `-xvis=no`
`-mno-vis2`	Same as `-xvis=no`
`-mno-vis3`	Same as `-xvis=no`
`-mtune=`*a*	Same as `-xtarget=`*a*
`-mvis`	Same as `-xvis`
`-mvis2`	Same as `-xvis`
`-mvis3`	Same as `-xvis`
`-no-canonical-prefixes`	Accept and ignore
`-no-fma`	Same as `-fma=none`
`-no-fmaf`	Same as `-fma=none`
`-no-trigraphs`	Same as `-xtrigraphs=no`
`-nodefaultlibs`	Same as `-xnolib` (C++ only)
`-pass-exit-codes`	Warn and ignore
`-pedantic-errors`	Same as `-pedantic`
`-pipe`	Warn and ignore
`-save-temps`	Same as `-keeptmp`
`-shared`	Same as `-G`
`-specs=`*t*	Warn and ignore
`-traditional`	Same as `-Xs` (C only)
`-trigraphs`	Same as `-xtrigraphs=yes`

Architecture and CPU Options

The GCC compiler uses platform specific flags to control the instruction set selection for the generated code. This is done through the `-march` option on x86 platforms, and `-mcpu` flag on SPARC platforms. On

x86 and SPARC platforms, selecting a value for the instruction set will also choose a corresponding value for the `-mtune` option which controls how the code is scheduled. For more information, see x86 Options.

The Oracle Developer Studio compilers use the following options to control this behavior:

- `-xtarget` – Select instruction set and code scheduling.
- `-xarch` – Select the instruction set only.
- `-xchip`/`-xcache` – Control the code scheduling.

For more information, see the *Oracle Developer Studio 12.6: C User's Guide*.

The GCC compilers on x86 have many `-m` options to select the specific instruction usage outside the compatible instruction set extensions.

The GCC compilers define CPP macros to identify which `-m` options are in effect. Oracle Developer Studio does not yet define these macros. The macros are defined to 1, unless stated.

TABLE 8 Architecture Options

`gcc` Option	Oracle Developer Studio Option
`-mxxx`, where `xxx` is an instruction set extension.	none
`-march=xxx` (x86) `-mcpu=xxx` (SPARC) Select instructions and code scheduling.	`-xtarget`
`-mtune=xxx` Generate code scheduling to optimize for this specific chip.	`-xchip` `-xcache`

For more information, see the following documentation references:

- x86 Options (`https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html#x86-Options`)
- SPARC Options (`https://gcc.gnu.org/onlinedocs/gcc/SPARC-Options.html#SPARC-Options`)
- `-xarch` and `-xtarget` options in *Oracle Developer Studio 12.6: C User's Guide*

TABLE 9 x86 `-xarch` and `-m` Options

GCC Option	Oracle Developer Studio Option	GCC Predefines
`-mpentium`	`-xarch=pentium`	
`-mpentiumpro`	`-xarch=pentium_pro`	
`-msse`	`-xarch=sse`	
`-msse2`	`-xarch=sse2`	
`-msse3`	`-xarch=sse3`	`__SSE3__`
`-mssse2`	`-xarch=ssse3`	
`-msse4.1`	`-xarch=sse4_1`	`__SSE4_1__ __SSSE3__`
`-msse4.2`	`-xarch=sse4_2`	`__SSE4_2__ __POPCNT__`
`-mpopcnt`	`-xarch=ss4_2`	`__POPCNT__`
`-mlzcnt`	`-xarch=avx2`	`__LZCNT__`
`-msse4`	`-xarch=sse4`	

`-mavx`	`-xarch=avx`	`__AVX__`
		`__XSAVE__`
		`__BIGGEST_ALIGNMENT__=32`

`-mpclmul`	`-xarch=aes`	`__PCLMUL__`

GCC Option	Oracle Developer Studio Option	GCC Predefines
-maes	-xarch=aes	__AES__
-mrdrnd	-xarch=avx_i	__RDRND__
-msgsbase	-xarch=avx_i	
-mf16c	-xarch=avx_i	__F16C__
-mbmi	-xarch=avx2	__BMI__
-mbmi2	-xarch=avx512	__BMI2__
-mavx2	-xarch=avx2	__AVX2__
-mavx512f	-xarch=avx512	__AVX512F__ __FP_FAST_FMA__ __FP_FAST_FMAF__ __BIGGEST_ALIGNMENT__=64
-mprefetchwt1	-xarch=avx512	__PREFETCHWT1__ __PREFCHW__

TABLE 10 x86 -march and -xtarget Options

GCC -march / -mcpu / -mtune	Oracle Developer Studio -xtarget	GCC Predefines
generic	-xtarget=generic	
native	-xtarget=native	
(older CPUs omitted)	...	...
sandybridge	-xtarget=sandybridge	___corei7_avx __corei7_avx__ __sandybridge __sandybridge__
ivybridge	-xtarget=ivybridge	
haswell	-xtarget=haswell	__core_avx2 __core_avx2__ __haswell __haswell__

TABLE 11 SPARC -m and -xarch Options

GCC Option	Oracle Developer Studio Option	Predefines
-mvis	-xarch=sparcvis	__VIS__ = __VIS = 0x100
-mvis2	-xarch=sparcvis2	__VIS__ = __VIS = 0x200
-mvis3	-xarch=sparcvis3	__VIS__ = __VIS = 0x300
-mfmaf	-xarch=sparcfmaf	__FP_FAST_FMA=1 __FP_FAST_FMAF=1

TABLE 12 SPARC -march and -xtarget Options

-mcpu / -mtune	Oracle Developer Studio Option	Predefines
native	-xtarget=native	

-mcpu / -mtune	Oracle Developer Studio Option	Predefines
(older CPUs omitted)	...	...
-mcpu=niagara4	-xtarget=T4	Same as -mvis3 -mfmaf
-mcpu=niagara7	-xtarget=T7	

Object File Compatibility

This section discusses features of object files that affect compatibility between Oracle Developer Studio and GCC.

Annotations

Some of the Oracle Developer Studio tools depend on additional structural information about the generated code emitted by the compilers. This information is called "annotations" and it is emitted into an ELF section named ".annotations". Some features of the Code Analyzer depend on annotations. Generation of this data can be controlled using the -xannotate option. The commands that use this information are binopt, code-analyzer, discover, collect, and uncover.

C++ Mangled Names

C++ compilers generate ELF symbols that have type information encoded in them. These are called "mangled names." The format for the mangled names is an implementation detail, so Oracle Developer Studio code compiled in the compat=5 (Sun ABI) mode will not intermix correctly with code compiled with g++. Using the Oracle Developer Studio C++ compiler in gnu modes results in compatible object files. For more information, see *Oracle Developer Studio 12.6: C++ User's Guide*.

Features Using Instrumentation

Compiler features that require instrumenting the object code also require extra libraries or object files to be included at link time. When using these features, you need to link your executable or library with the Oracle Developer Studio compiler. Note that mixing gcc-built code into your program or library will not be instrumented, so the results would be incomplete.

Features in this category are:

- Profile Feedback – Includes the options -xprofile and -xlinkopt.
- Traditional Profiling – Includes the option -xpg at compile and link time.
- Traditional Coverage – Includes the option -xprofile=tcov and the tcov utility.

Debug Information

dbx and other Oracle Developer Studio tools use debugging information generated by the compiler. The majority of the information is recorded in the DWARF format, but some additional indexing information is recorded in an older format called STABS. The ELF sections for dwarf start with ".debug" and the ELF sections for stabs start with ".stab".

Runtime Support Compatibility

This section discusses issues related to runtime support.

`libgcc` Support

Compilers will automatically link with `libgcc` when necessary, but if something goes wrong, knowing what the library is designed to do can be helpful. The `gcc` support library (`libgcc`) has a variety of helper routines in the form of an archive library called `libgcc.a` and a shared library called `libgcc_s.so`. `libgcc` includes routines to support exceptions and routines to support integer and floating-point operations that might not have direct instructions for the current architecture. The functionality in the library is described in The GCC Low-level Runtime Library in the `gcc` documentation.

Executables created using the `gcc` compiler are linked against `libgcc.a` and will not have a dynamic dependency against `libgcc_s.so`. Executables created with the `g++` compiler will use `libgcc_s.so` instead and will have a dynamic dependency on it.

Code produced by the Oracle Developer Studio C compiler does not have any dependencies on `libgcc`.

Code produced by the Oracle Developer Studio C++ compiler does not depend on `libgcc` when building in Sun mode (`-compat=5`), but will have such a dependency when building in GNU compatibility mode (`-compat=g`, `-std=c++03`, `-std=c++11`, or `-std=c++14`).

On Oracle Linux, a similar library named `libc_supp.a` is included with Oracle Developer Studio.

C++ Runtime Support

A source file compiled with the Oracle Developer Studio C++ compiler needs to be linked using the Oracle Developer Studio C++ compiler to make sure that the correct runtime libraries, CRT files, and linker options are used. `g++` object files should be linked with `g++`. Because of the many incompatible implementation details used in object files, mixing object files from the Oracle Developer Studio C++ and `g++` compilers into the same executable or shared library is not supported.

GNU ABI Compatibility

Shared libraries created by Oracle Developer Studio C++ in GNU compatibility mode (with the options `-compat=g`, `-std=c++03`, `-std=c++11`, or `-std=c++14`) can be mixed with shared libraries created by the `g++` compiler and linked into a main program created by either compilers, but they must use the same version of the `g++` ABI.

When operating in GNU compatibility mode, the Oracle Developer Studio C++ compiler is binary compatible with `g++` code in the area of library interfaces, name mangling, and the binary layout of standard library objects. But it still uses different mechanisms for many other underlying aspects of the implementation. External inline functions, template instances, RTTI records, and exception range information are implemented using COMDAT in a distinct way. Exception range information is formatted in a very Studio-specific way. This limits the amount of mixing you can do between object files and archive libraries compiled with Oracle Developer Studio C++ and those compiled with `g++`.

GNU C++ ABI 4.*x* to 5.*x* Change

Due to an incompatible change in the GNU C++ ABI between versions 4.*x* and 5.*x*, executables and libraries compiled with `g++` must use one ABI consistently to work correctly. The 5.*x* GNU C++ library supports both ABIs, but a compiler (GCC or Oracle Developer Studio) can generate only one ABI at a time. The `g++` compiler defaults to the 5.*x* ABI. To select the 4.*x* ABI, compile with `-D_GLIBCXX_USE_CXX11_ABI=0`.

The Oracle Developer Studio 12.6 C++ compiler implements only the 4.*x* ABI. To mix shared libraries and executables, build the `g++` parts with the above option. The `g++` documentation for this issue is available at `https://gcc.gnu.org/onlinedocs/libstdc++/manual/using_dual_abi.html`.

Technically, there is nothing to stop a 4.*x* library or executable from being used alongside a 5.*x* library or executable, but they cannot call each other using library classes that are different in the two ABIs. Some of those classes are commonly used (like the string class), so you cannot accomplish much by mixing the two ABIs.

`main` function

The `main` function in C++ carries some extra mechanisms. Whichever compiler is used to compile the object file with `main` should also be used to link the executable. The `crt` object files used by the compiler driver will match the mechanism embedded in the object file `main`.

`libCrunG3.so` Library

When compiling code in GNU mode, the Oracle Developer Studio C++ compiler uses the GNU C++ library, but it also needs to link an additional library called `libCrunG3.so` to supply support functions for RTTI, exceptions, and other language features. This library corresponds to the `libCrun.so` library that is used in Sun mode (`-compat=5`).

Different Versions of the GNU C++ Library

The Oracle Developer Studio 12.6 C++ compiler (in GNU compatibility mode) uses the 5.4.*x* version of the GNU C++ library. This library supports both the 4.x and 5.x `g++` ABIs. For more information, see “GNU ABI Compatibility” on page 19. You can mix shared libraries and applications built with different versions of the GNU C++ library, but the compiler used to build and link the main application should be a version at least as recent as the compiler used to create any of the shared libraries. If any shared libraries are built with Oracle Developer Studio C++, the main application should be built either with Oracle Developer Studio C++ or with a `g++` compiler that supports the 5.4.*x* GNU C++ library.

Runtime Library Locations

C++ runtime libraries are available at different locations.

The following Sun mode (`-compat=5`) C++ libraries are in `/usr/lib` on Oracle Solaris, but not on Oracle Linux:

- `libCstd.so`
- `libCrun.so`
- `libdemangle.so`

- `libiostream.so`
- `libstdcxx.so`

The following GNU mode C++ libraries are in `/usr/lib` on Oracle Solaris, but not on Oracle Linux:

- `libCrunG3.so` (but you might need a patch for Oracle Solaris package `SUNWlibC`)
- `libdemangle.so`

All other C++ runtime libraries are in the compiler installation directory.

When the Oracle Developer Studio C++ compiler links against any shared libraries that are bundled with Oracle Developer Studio, it normally adds the necessary `RPATH` setting into the executable to find the libraries at runtime. You can prevent this behavior and configure the `RPATH` setting yourself by adding the `-xnorunpath` option and appropriate `-R` options.

The Atomics runtime support library is new in Oracle Developer Studio 12.5 and is required by default with C++. For more information, see "Atomics" on page 21 section.

OpenMP

Compiling source code with `-xopenmp` generates runtime dependencies on system libraries, which for the Oracle Developer Studio compilers is the `libmtsk` library. On Oracle Solaris, this library is available in `/usr/lib`. On Oracle Linux, this library is in the compiler installation directory. Since thread management is a process-wide operation, use only one OpenMP implementation in a program. If a shared library uses OpenMP, the main application cannot use a different OpenMP implementation. The Oracle Developer Studio implementation of OpenMP is not compatible with the GCC implementation of OpenMP.

Atomics

Atomics is a new language feature in the C 2011 and C++ 2011 standards that requires runtime support from the operating system. The Oracle Developer Studio compilers support this feature with a preliminary bundled runtime library that is compatible with some versions of GCC runtime libraries.You can use the `-xatomic` option to select which runtime library to use when linking your programs and libraries.

For more information about atomics and the runtime support, see "Bundled Atomics Library" in *Oracle Developer Studio 12.6: C User's Guide* and "Bundled Atomics Library" in *Oracle Developer Studio 12.6: C++ User's Guide*

C++ applications built in GNU mode or in C++ 11 mode will require an atomic support library, either from Oracle Developer Studio or GCC. When running C++ applications that were built with the `-xnorunpath` option, you might get the following error:

```
ld.so.1: a.out: fatal: libstatomic.so.1: open failed: No such file or directory
```

You can resolve this by explicitly linking with the `gcc` version of atomics support, or by copying the Oracle Developer Studio support library and including it in your application.

Thread Local

The GCC and Oracle Developer Studio compilers implement thread local data (`__thread declarations`) in a way that conforms to the OS ABI (Oracle Solaris or Oracle Linux). So, functionality is compatible

between code compiled by Oracle Developer Studio and GCC compilers. For more information about the Oracle Solaris ABI for thread local data, see the *Oracle Solaris 11.3 Linkers and Libraries Guide*.

Shared Library Compatibility

A source file compiled with Oracle Developer Studio C++ compiler (CC) needs to be linked using the C++ compiler to make sure if the correct runtime libraries and linker options are used. G++ object files should be linked with the `g++` compiler.

Shared libraries created by Oracle Developer Studio CC with the options `-compat=g`, `-std=c++03`, `-std=c++11`, or `-std=c++14` can be freely mixed with shared libraries created by the `g++` compiler and linked into a main program created by either of these compilers. A binary compiled against a newer version of the `gcc` library in most instances cannot be linked to an older version of the `gcc` library.

The Oracle Developer Studio 12.6 C++ compiler in `gcc` compatible mode uses the 5.4.*x* version of the `g++` runtime library.

You can mix libraries and applications built with different versions of the `g++` library, but the compiler used to build and link the main application should be a version at least as recent as the compiler used to create any of the shared libraries. If any shared libraries are built with Oracle Developer Studio C++, the main application should be built either with the Oracle Developer Studio C++ or with a `g++` compiler that supports the 5.4.*x* runtime libraries.

Tool Compatibility

This section discusses the compatibility of Oracle Developer Studio tools with GCC.

dbx Debugger

`dbx` can debug C and C++ code generated by both Oracle Developer Studio and GCC compilers. It is compatible with the versions of GCC listed in "GCC Version Compatibility" on page 23. For more information about `dbx`, see *Oracle Developer Studio 12.6: Debugging a Program with dbx*.

Performance Analyzer

Performance Analyzer supports mixed Oracle Developer Studio and GCC binaries. It can analyze code compiled by Oracle Developer Studio and GCC compilers and supports versions of GCC listed in "GCC Version Compatibility" on page 23. For more information about Performance Analyzer, see *Oracle Developer Studio 12.6: Performance Analyzer*.

Code Analyzer

The `discover` ADI (`-i` *adi*) and `discover` Lite (`-l`) options support Oracle Developer Studio and `gcc` binaries. The `discover(1)` utility detects errors in Oracle Developer Studio compiled code but does not check `gcc` compiled code. The `uncover(1)` utility gathers coverage information for Oracle Developer Studio compiled code. For more information about the code analytic tools, see *Oracle Developer Studio 12.6: Code Analyzer User's Guide* and *Oracle Developer Studio 12.6: Discover and Uncover User's Guide*.

IDE

An IDE project supports code written or built with only one compiler, either GCC or Oracle Developer Studio. The workaround for this limitation is to create separate projects for GCC-written code.

GCC Version Compatibility

GCC versions 4.8.2, 4.9.2 and 5.1.0 are supported on the following operating systems:

- Solaris 10 1/13 (Update 11)
- Oracle Solaris 11.2
- Oracle Solaris 11.3
- Oracle Linux 6.6
- Oracle Linux 7.*x*

GCC version 4.4 is supported on Oracle Linux 6.6.

References for More Information

- Extensions to the C Language Family
- Extensions to the C++ Language
- GCC Binary Compatibility
- GNU C++ ABI Policy and Guidelines
- Intel® Compilers for Linux: Compatibility with GNU Compilers

Oracle Developer Studio 12.6: GCC Compatibility Guide

Part No: E77792

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.

Référence: E77792

Accès aux services de support Oracle

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.