



Intel® Compilers for Linux*

Compatibility with GNU Compilers

Introduction

This paper describes the compatibility between Intel® Compilers for Linux* and GNU compilers in terms of source, binary, and command-line compatibility. The Intel® C++ and Fortran Compilers help make your software run at top speeds on Intel platforms, including IA-32, processors with Intel® EM64T, and Intel® Itanium® processor-based systems. The compilers also provide compatibility with commonly-used Linux software-development tools.

Overview

Intel C++ Compiler 9.0 for Linux provides excellent source and binary compatibility with the GNU GCC compiler. Compared to previous versions, Intel C++ Compiler 9.0 has improved compiler-option compatibility with GCC and general improvements related to the GCC build environment.

The motivation for improved compatibility comes from customer requests in several areas:

- Mixing and matching binary files created by g++, including third-party libraries
- Generating C++ code compatible with g++ 3.2, 3.3 or 3.4
- Building the Linux kernel with the Intel Compiler with fewer workarounds
- Improved support for command-line options offered in the GNU compilers

Intel C++ Compiler for Linux supports the ANSI C and C++ standards, most GNU C and C++ language extensions, and the OpenMP* 2.5 standards. Intel C++ Compiler for Linux is binary or object-file compatible with C-language binary files created with the GNU GCC and C++ binary compatibility with g++ versions 3.2, 3.3, and 3.4 (see the *Binary Compatibility* section of this document).

Building the Linux kernel with Intel C++ Compiler is an ongoing project at Intel. The goal is to improve GCC source compatibility of Intel C++ Compiler, and to find opportunities to improve kernel performance. Intel Corporation and Red Flag Software Co., Ltd, announced that Red Flag is the first company to use the Intel C++ Compiler for Linux to compile a

commercial version of its Linux operating system. Details of this announcement are available at <http://www.intel.com/pressroom/archive/releases/20040803net.htm>.

Intel® Fortran Compiler 9.0 for Linux supports the Fortran95 and OpenMP 2.5 standards. Intel Fortran Compiler is not binary compatible with GNU g77 compiler, nor is binary compatibility with g77 a future goal of the Fortran compiler. Intel Fortran Compiler is binary compatible with C-language object files created with either Intel C++ Compiler or the GNU GCC compiler.

Intel C++ and Fortran Compilers support a large number of the commonly used GNU compiler command-line options. See the *Command Line Options* section of this document for details, including the large number of options for which support has recently been added.

We encourage customers to submit feature requests for additional compatibility through their Intel® Premier Support accounts. See *Customer Feedback* below.

C and C++ Source Compatibility

Intel C++ Compiler supports the ANSI C and C++ standards. The *Intel® C++ Compiler for Linux User's Guide* provides information on C99 features that are supported in version 9.0. Intel C++ Compiler also supports most GNU C and C++ language extensions. Excellent source compatibility allows the definition of the GCC predefined macros:

- `__GNUC__`
- `__GNUC_MINOR__`
- `__GNUC_PATCHLEVEL__`

The definition of these macros greatly reduces the amount of compiler-specific code, including the system header files. These macro definitions can be disabled using the **-no-gcc** option.

GNU inline-assembly format has been supported on IA-32 processors since Intel C++ Compiler 6.0 for Linux. The Intel Compiler also supports GNU inline-assembly for Intel processors with Intel EM64T. The Intel C++ Compiler provides intrinsic functions for the Itanium

2 processor that provide equivalent inline assembly functionality without inhibiting compiler optimizations important on Itanium processor-based systems. Intel C++ Compiler implements a large number of GCC built-in functions, which are documented at <http://gcc.gnu.org/onlinedocs/gcc/Other-Builtins.html>. See the User's Guide for information on intrinsic functions and supported GCC built-in functions.

Table 1 summarizes GNU C-language extensions supported by Intel C++ Compiler 5.0, 6.0, 7.0, 8.1, and 9.0, highlighting continuous source-compatibility improvements. Table 2 shows supported GNU C++ language extensions. Note that some C extensions are not supported when compiling C++ source files. The GCC C and C++ language extensions are documented in the GCC manual, available at <http://gcc.gnu.org>.

Linux* Kernel Build

Intel C++ Compiler 9.0 for Linux has excellent support for GNU C-language extensions. As a demonstration of compatibility with GCC, Intel C++ Compiler for Linux has successfully built and run Linux kernels on IA-32 and Itanium processors using a limited number of temporary source patches. The temporary nature of these patches will be addressed in the future, either by adding extensions to the Intel C++ Compiler or by working with the Linux community to replace or reduce usage of less frequently used, non-standard language features in the kernel sources.

Intel Corporation and Red Flag Software Co., Ltd, announced that Red Flag is the first company to use the Intel® C++ Compiler for Linux to compile a commercial version of its Linux operating system. Details of this announcement are available at <http://www.intel.com/pressroom/archive/releases/20040803net.htm>.

Binary Compatibility

Intel C++ Compiler 9.0 provides C and C++ binary compatibility with GCC 3.2, 3.3, and 3.4. Since Release 8.1, the default C++ library implementation is the GCC-provided C++ libraries, equivalent to the **-cxxlib-gcc** option, on systems with a compatible GCC version. On systems without a compatible GCC version, the C++ libraries provided by Intel are used. This is equivalent to the **-cxxlib-icc** option, which does not provide C++ binary compatibility with GCC due to differences in the C++ library implementation.

Intel C++ Compiler 9.0 for Linux supports the C++ ABI*, a convention for binary object code interfaces between C++ code and the implementation-provided system and runtime libraries. The GCC 3.x compilers also conform to the C++ ABI, allowing Intel C++ Compiler 9.0 to be binary compatible with g++ 3.2, 3.3, or 3.4, with slight differences to account for minor ABI changes in g++. Consult <http://gcc.gnu.org/releases.html> for information on the latest GCC releases. The goal is the implementation of a stable ABI for C++ applications and libraries, which is a benefit to the Linux community.

The Intel C++ Compiler 9.0 **-cxxlib-gcc** option allows developers to build applications using the C++ runtime provided by GCC. The GCC C++ runtime includes the **libstdc++** standard C++ header files, library, and language support. Unless the **-cxxlib-icc** option is specified, these GCC components are used instead of the Intel-provided header files and libraries, including **libcprts** and **libcxa**. On processors with Intel EM64T, only the GCC C++ runtime libraries are supported, allowing the Intel Compiler to always generate g++ binary compatible code.

With the **-cxxlib-gcc** option, the resulting C++ object files, libraries, and executables can interoperate with C++ object files, libraries, and executables generated by supported GCC versions 3.2, 3.3, or 3.4. This means that third-party C++ libraries built with supported GCC versions will work with C++ code generated by the Intel Compiler.

On systems with supported GCC versions, the default code generation is compatible with the GCC version in your **PATH** environment variable. It is recommended to set **-cxxlib-gcc** and **-gcc-version** options explicitly to avoid accidental errors due to changing the GCC version specified by your **PATH** environment variable. The **-gcc-version=<version>** option allows great flexibility to generate code for the supported GCC versions:

- **-gcc-version=320** for GCC 3.2 compatibility
- **-gcc-version=330** for GCC 3.3 compatibility
- **-gcc-version=340** for GCC 3.4 compatibility

Table 1. GCC C-Language Extensions Supported in the Intel® C++ Compiler for Linux*

GCC C Language Extension	Intel® C++ Compiler 5.0	Intel C++ Compiler 6.0	Intel C++ Compiler 7.0	Intel C++ Compiler 8.1	Intel C++ Compiler 9.0
Statement Expressions	No	Yes	Yes	Yes	Yes
Locally Declared Labels	No	No	Yes	Yes	Yes
Labels as Values	No	Yes	Yes	Yes	Yes
Nested Functions	No	No	No	No	No
Constructing Function Calls	No	No	No	No	No
Naming an Expression's Type	No	No	Yes	Yes	Yes
Referring to a Type with typeof	No	Yes	Yes	Yes	Yes
Generalized Lvalues	No	No	Yes	Yes	Yes
Conditionals Omitted Operands	No	Yes	Yes	Yes	Yes
Double Word Integers	No	No	Yes	Yes	Yes
Complex Numbers	No	Yes	Yes	Yes	Yes
Hex Floats	No	No	Yes	Yes	Yes
Arrays of Zero Length	No	No	Yes	Yes	Yes
Arrays of Variable Length	No	No	Yes	Yes	Yes
Macros with Variable Number of Arguments	No	Yes	Yes	Yes	Yes
Looser Rules for Escaped Newlines	No	No	No	No	No
Strings Literals with Embedded Newlines	No	No	Yes	Yes	Yes
Non-Lvalue Arrays Subscripts	No	No	Yes	Yes	Yes
Arithmetic on Void Pointers	No	Yes	Yes	Yes	Yes
Arithmetic on Function Pointers	No	No	No	No	Yes
Non-Constant Initializers	No	No	Yes	Yes	Yes
Compound Literals	No	No	Yes	Yes	Yes
Designated Initializers	No	Yes	Yes	Yes	Yes
Cast to Union Type	No	No	Yes	Yes	Yes
Case Ranges	No	Yes	Yes	Yes	Yes
Mixed Declarations and Code	No	No	Yes	Yes	Yes
Function Attributes	Few	Few	Most	Most	Yes
Prototype and Old-Style Function Definitions	No	No	No	No	No
C++ Style Comments	Yes	Yes	Yes	Yes	Yes
Dollar Sign in Identifier Names	No	No	Yes	Yes	Yes
The ESC Character in Constants	No	No	Yes	Yes	Yes
__alignof__ (types, variables)	No	Yes	Yes	Yes	Yes
Attributes of Variables	Few	Few	Most	Most	Yes
Inline Function as Fast as a Macro	Yes	Yes	Yes	Yes	Yes
Inline ASM (IA-32)	No	Yes	Yes	Yes	Yes
Controlling Names Used in ASM Code	No	No	Yes	Yes	Yes
Variables in Specified Registers	No	No	Yes	Yes	Yes
Alternate Keywords	No	No	Yes	Yes	Yes
Incomplete enum Types	No	Yes	Yes	Yes	Yes
Function Name as Strings	No	No	Yes	Yes	Yes
Getting the Return or Frame Address of a Function (IA-32)	No	No	Yes	Yes	Yes
Other Built-in Functions	No	No	Few	Most	Yes
Using Vector Instructions Through Built-In Functions	No	No	No	No	No
Built-in Functions Specific to Particular Target Machines	No	No	No	No	No
Pragmas Accepted by GCC	No	No	No	No	No

Table 2. GCC C++ Language Extensions Supported in the Intel® C++ Compiler for Linux*

GCC C++ Language Extensions	Intel® C++ Compiler 9.0
Minimum and Maximum Operators in C++	Yes
When Is a Volatile Object Accessed?	No
Restricting Pointer Aliasing	Yes
Vague Linkage	Yes
Declarations and Definitions in One Header	No
Where's the Template?	External Template Supported
Extracting the Function Pointer from a Bound Pointer to Member Function	No
C++-Specific Variable, Function, and Type Attributes	Yes
Java* Exceptions	No
Deprecated Features	No
Backwards Compatibility	No

Future product versions may include compatibility options for more recent versions of g++. Contact Intel through your Intel Premier Support account for additional information.

The following examples demonstrate usage of the different C++ runtime libraries. Figure 1 uses the default **-cxxlib-gcc** C++ binary-compatibility option to use the g++ provided C++ runtime libraries. Note that the Intel C++ runtime libraries are not linked to the application, although the **libcxaguard** library is used for g++ interoperability support. The g++ libraries are the standard GNU C++ library, **libstdc++**, and C++ language support library, **libgcc_s**.

The system utility **ldd** is used to show the dynamic libraries linked to the application, and the **awk** utility is used to make the output easier to read in this paper.

```
prompt> cat hello.cxx
#include <iostream>
int main(){ std::cout<<"Hi"<<std::endl;
}

prompt> icpc hello.cxx -cxxlib-gcc
prompt> ldd a.out | awk '{print $1}'
libm.so.6
libstdc++.so.5
libgcc_s.so.1
libcxaguard.so.5
libc.so.6
/lib/ld-linux.so.2
```

Figure 1. Example Using the GCC C++ Runtime Libraries

Figure 2 illustrates creating an application with the C++ runtime libraries supplied by Intel and shows the application linked to the C++ libraries provided by Intel:

- Intel C++ library **libcprts**
- Intel C++ language-supported libraries **libcxa** and **libunwind**
- Intel C++ Compiler for Itanium processors also links to **libipr** for g++ interoperability support.

```
prompt> icpc hello.cxx -cxxlib-icc
prompt> ldd a.out | awk '{print $1}'
libimf.so
libm.so.6
libcprts.so.5
libcxa.so.5
libunwind.so.5
libc.so.6
/lib/ld-linux.so.2
```

Figure 2. Example Using C++ Runtime Libraries Provided by Intel

The example in Figure 3 shows mixing binary files created by g++ and Intel C++ Compiler 9.0 with the **-cxxlib-gcc** C++ binary-compatibility option.

```
prompt> cat main.cxx
void pHello();
int main() { pHello(); }

prompt> cat pHello.cxx
#include <iostream>
void pHello()
{ std::cout << "Hello" << std::endl; }

prompt> icpc -c main.cxx -cxxlib-gcc
prompt> g++ -c pHello.cxx
prompt> icpc main.o pHello.o -cxxlib-gcc
prompt> ./a.out
Hello
```

Figure 3. Mixing Binary Files Created by g++ and Intel® C++ Compiler 9.0

Figure 4 shows how to mix binary files created by g++ and Intel C++ Compiler 9.0, using g++ to link. Intel recommends linking with the Intel Compiler to correctly pass the Intel libraries to the system linker, ID. This example shows how to use g++ for linking. On IA-32 and processors with Intel EM64T, Intel C++ Compiler calls the function **intel_proc_init** from the main routine to determine the ability to run processor-specific code.

This routine is found in the Intel library **libirc**. To link this example correctly, **libirc** needs to be added to the link command. The compiler for Itanium architecture requires linking in the additional library **libipr** via the option **-lipr** for g++ interoperability support. Different compiler optimizations, such as OpenMP and vectorization, may require additional libraries available from Intel.

Linking with the Intel Compiler removes the necessity of knowing the details of which Intel libraries are required, provided the same compiler options are used when compiling and linking. Without passing the correct libraries and library location, the initial link fails.

For the example in Figure 4, linking with the Intel **libirc** library is required.

```
prompt> icpc -c main.cxx -cxxlib-gcc
prompt> g++ -c pHello.cxx

# Fails without Intel provided libraries
prompt> g++ main.o pHello.o
main.o: In function `main':
main.o(.text+0xd): undefined reference to
`__intel_proc_init'
collect2: ld returned 1 exit status

# Links using Intel provided libraries.
prompt> g++ main.o pHello.o -L /opt/intel/
cc/9.0/lib -lirc -lcxaguard
prompt> ./a.out
Hello
```

Figure 4. Example of Linking Using g++

Linking and Libraries

Linking C Language with Intel Compiler and GCC Compiler

C-language object files can be linked with either Intel Compilers or GCC compilers. Linking with the Intel Compiler is recommended, as the Intel libraries will be correctly passed to the linker. The example shown in Figures 5 through 10 uses GCC to link the application with the file **main.c**, compiled with GCC, and the file **calcSin.c**, compiled with the Intel C++ Compiler for IA-32 applications and optimized for the Intel® Pentium® 4 processor. The paper, *Optimizing Applications with the Intel® C++ and Fortran Compilers*, available at <http://www.intel.com/software/products/compilers/clin>, describes advanced compiler optimizations available with the Intel Compiler.

Figure 5 uses the Intel Compiler to compile the **calcSin.c** function and is able to automatically vectorize the loop.

```
prompt> cat calcSin.c
#include <math.h>
void calcSin(double *a, double *b, int N) {
    int i;
    for (i=0; i<N; i++)
        b[i] = sin(a[i]); }

prompt> icc -c -xP calcSin.c
remark: LOOP WAS VECTORIZED.
```

Figure 5. Compiling C Language Function with the Intel® C++ Compiler

The main function is compiled with GCC in Figure 6.

```
prompt> cat main.c
void calcSin(double *a,double *b,int N);
int main() {
    const int N=100000;
    double a[N], b[N], c[N], x[N];
    int i;
    for (i=0;i<N;i++)
        a[i] = i;
    for (i=0;i<100;i++)
        calcSin( a, b, N); }
```

Figure 6. Main Function Compiled with GCC

The application is linked with GCC, and the necessary libraries from the Intel Compiler are passed to GCC. In the example in Figure 7, the short vector math library, **libsvml.a**, and the Intel provided math function library, **libimf.a**, are required to link.

```
prompt> gcc main.o calcSin.o -L /opt/
intel/cc/9.0/lib -lsvml -limf -o calcSin
```

Figure 7. Linking libsvml.a and libimf.a

The **nm** utility can determine the location of unresolved symbols in the Intel libraries. If the Intel libraries are not passed to GCC, the link fails with undefined references, shown in Figure 8.

```
prompt> gcc main.o calcSin.o
calcSin.o(.text+0x51): undefined reference
to `__libm_sse2_sin'
calcSin.o(.text+0x76): undefined reference
to `vmlDsin2'
calcSin.o(.text+0xa0): undefined reference
to `__libm_sse2_sin'
```

Figure 8. Failed Link Due to Missing Libraries When Linking With GCC

Running the **nm** utility on the Intel libraries helps determine which libraries are required. See the Intel C++ and Fortran User's Guides, library section, for documentation on the different Intel libraries. Next, the **grep** utility (Figure 9) verifies that the symbols are defined in the Intel libraries. Examine the symbol file, **icc-symbols.txt**, to determine which library contains the missing symbol and add this to the link options.

```
prompt> nm /opt/intel/cc/9.0/lib/* > icc-
symbols.txt
prompt> grep `__libm_sse2_sin` icc-
symbols.txt
000b73f0 T __libm_sse2_sin
```

Figure 9. Utilities to Determine Libraries Required to Link with GCC Correctly

Link with the Intel Compiler, passing the same options (-xP in Figure 10) used during compilation, to avoid the necessity of finding out what Intel supplied libraries are required.

```
prompt> gcc -c main.c
prompt> icc -c -xP calcSin.c
prompt> icc -xP calcSin.o main.o
```

Figure 10. Avoiding Missing Libraries When Linking by Using Intel® C++ Compiler with the Same Options as Used During Compilation

Intel® Compilers Linking Conventions

This section describes conventions used by Intel Compilers for Linux and compiler options for changing the default behavior. The **icc** compiler driver, for the C language, does not link the C++ runtime libraries when compiling C applications. The **icc** compiler driver links the C++ runtime libraries if the input source files have C++ file extensions. Previous **icc** versions always linked against the C++ runtime libraries. The changes in the **icc** driver were made to be more consistent with the **GCC** compiler. The **icpc** compiler driver is meant to be used for C++ files, and it automatically links in the C++ runtime libraries, similar to the **g++** behavior.

By default, Intel Compilers for Linux use Dynamic Shared Object (DSO) versions, also known as shared libraries, of the Linux system libraries. For the Intel provided libraries, by default the DSO version of `libcxaguard`, the g++ compatibility support library, is used and static versions of all other Intel libraries are used. For user-provided libraries, a DSO version is searched for first. If no DSO version is found, a static version is searched for.

The following command line options modify the default behavior:

- **-static** Link in all libraries statically, and create a statically linked executable.
- **-Bstatic**: Use the static version of all libraries specified after this point or until the option **-Bdynamic** is used. This option can be used to statically link all libraries.
- **-Bdynamic**: Use dynamic (DSO) version of all libraries specified after this point or until the **-Bstatic** option is used. Note that **-Bstatic** and **-Bdynamic** are toggles.
- **-i-static**: Statically link all compiler libraries provided by Intel. This option can avoid the need to redistribute the libraries with your application.
- **-i-dynamic**: Dynamically link all compiler libraries provided by Intel. In other words, use DSO versions of Intel libraries.
- **-shared**: Instructs the linker to create a DSO instead of an application binary.

Note that **-shared** is not the opposite of the **-static** option.

The following options provide additional flexibility:

- **-static-libcxa**: Link the **libcxa** library statically. This option conflicts with the C++ ABI and is the reason libraries provided by Intel are linked dynamically by default.
- **-dynamic-libcxa**: Link the **libcxa** library dynamically.

The **ldd** utility lists the DSOs that an application is linked with and is useful in understanding the command-line options described previously.

Optimized Math Function Library

The math function library, **libimf.a**, is an optimized math library provided with the Intel Compilers for Linux. The default Linux math library, **libm**, contains functions in addition to those provided in the math function library. The Intel Compilers first link to the **libimf** library, then to the **libm** library. If an optimized function is available, it will be found in the math-function library, **libimf**. Otherwise, it is linked from the Linux math library, **libm**.

Note: New in version 9, the compiler always links to functions in the optimized **libimf** library if available, regardless of whether the user specifies **-lm** on the link command line or if the user adds both **libimf** and **libm**, and regardless of the order of math libraries.

Build Environment Enhancements

Intel C++ Compiler continues to improve compatibility with the GCC build environments. New in Version 9 are improvements in the ability to use the g++ C++ library on systems with non-standard GCC configurations, and as a result, customers have reported improved usability on Debian Linux distributions, although this distribution is not officially supported.

Version 9 now supports the **GCC GCC_EXEC_PREFIX** environment variable, which allows the specification alternative names of the linker (**ld**) and assembler (**as**), which also allows changing the directory for **ld** and **as**.

Command-Line Options

Intel C++ Compiler for Linux supports a large number of common GNU compiler command-line options. Table 3 lists the GCC compiler options for which support has been recently added in the Intel C++ Compiler for Linux. Due to the large number of supported options, the full list of supported options is not included in this document. Information on Intel C++ command options can be found in the User's Guide, compiler man pages, and summary information via **icc -help**.

The meaning of the **-ansi** switch was changed starting with Intel C++ Compiler 8.1, to be compatible with the GCC command-line option of the same name. The Intel Compiler can support stricter conformance of semantics to ISO C and C++. This support is implemented by using the **-strict-ansi** command-line option.

Table 3. Recently Added GCC Compiler Command-Line Options for Intel® Compilers for Linux*

GCC Compiler Command Linux* Option	Description
<code>-fargument-alias</code>	same as <code>-alias-args</code>
<code>-fargument-noalias</code>	same as <code>-alias-args-</code>
<code>-fargument-noalias-global</code>	arguments do not alias each other and do not alias global storage
<code>-fdata-sections</code>	same as <code>-ffunction-sections</code>
<code>-f[no-]exceptions</code>	enable(DEFAULT)/disable exception handling
<code>-ffunction-sections</code>	separate functions for the linker (COMDAT)
<code>-finline-functions</code>	inline any function, at the compiler's discretion (same as <code>-ip</code>)
<code>-f[no-]math-errno</code>	set ERRNO after calling standard math library functions
<code>-fno-builtin</code>	disable inline expansion of intrinsic functions
<code>-fno-builtin-<func></code>	disable the <code><func></code> intrinsic
<code>-fno-gnu-keywords</code>	do not recognize 'typeof' as a keyword
<code>-fno-operator-names</code>	disable support for operator name keywords
<code>-f[no-]omit-frame-pointer</code>	negative version same as <code>-fp</code>
<code>-fpack-struct</code>	pack structure members together
<code>-fpermissive</code>	allow for non-conformant code
<code>-freg-struct-return</code>	return struct and union values in registers when possible
<code>-ftemplate-depth-<n></code>	control the depth in which recursive templates are expanded
<code>-funroll-loops</code>	unroll loops based on default heuristics
<code>-I-</code>	any directories you specify with '-I' options before the '-I-' options are searched only for the case of '#include "FILE"'; they are not searched for '#include <FILE>'
<code>-imacros <file></code>	treat <code><file></code> as an <code>#include</code> file, but throw away all preprocessing while macros defined remain defined
<code>-iprefix <prefix></code>	use <code><prefix></code> with <code>-iwithprefix</code> as a prefix
<code>-iwithprefix <dir></code>	append <code><dir></code> to the prefix passed in by <code>-iprefix</code> and put it on the include search path at the end of the include directories
<code>-iwithprefixbefore <dir></code>	similar to <code>-iwithprefix</code> except the include directory is placed in the same place as <code>-I</code> command line include directories
<code>-malign-double</code>	same as <code>-align</code>
<code>-m[no-]ieee-fp</code>	same as <code>-mp</code>
<code>-msse</code>	generate code for Intel® Pentium® III and compatible Intel processors
<code>-msse2</code>	generate code for Pentium 4 and compatible Intel processors
<code>-msse3</code>	generate code for Pentium 4 processors with SSE3 extensions
<code>-mtune=<cpu></code>	optimize for a specific CPU
<code>-W[no-]abi</code>	warn if generated code is not C++ ABI compliant (DEFAULT)
<code>-Wcontext-limit=<n></code>	set maximum number of template instantiation contexts shown in diagnostic
<code>-W[no-]deprecated</code>	print warnings related to deprecated features
<code>-Winline</code>	enable inline diagnostics
<code>-W[no-]comment[s]</code>	warn when <code>/*</code> appears in the middle of a <code>/* */</code> comment
<code>-W[no-]main</code>	warn if return type of <code>main</code> is not expected

continued on page 10

Table 3. Recently Added GCC Compiler Command-Line Options for Intel® Compilers for Linux* (continued)

GCC Compiler Command Linux* Option	Description
-W[no-]missing-prototypes	warn for missing prototypes
-W[no-]pointer-arith	warn for questionable pointer arithmetic
-W[no-]return-type	warn when a function uses the default int return type and warn when a return statement is used in a void function
-W[no-]uninitialized	warn if a variable is used before being initialized
-W[no-]unknown-pragmas	warn if an unknown #pragma directive is used (DEFAULT)
-W[no-]unused-function	warn if declared function is not used
--version	display GCC style version information

Note that Intel C++ and Fortran Compilers have a large number of features to optimize applications for the latest IA-32 processors and Itanium processor-based systems. The paper “Optimizing Applications with the Intel C++ and Fortran Compilers,” available at <http://www.intel.com/software/products/compilers/clin>, explains how to use Intel Compilers to optimize for Pentium 4 and Itanium processors.

Intel encourages customers to submit feature requests for command-option compatibility through their Intel Premier Support account. See Customer Feedback section in this document for more information.

Intel® Fortran Compiler

Intel Fortran Compiler supports the Fortran95 and OpenMP 2.5 standards. Intel Fortran Compiler for Linux is not binary compatible with the GNU g77 compiler, nor is this a future goal. In general, Fortran compilers are not binary compatible, due to using different runtime libraries. Intel Fortran Compiler is binary compatible with C-language object files created with either Intel C++ Compiler for Linux or the GNU GCC compiler. The *Intel Fortran Compiler for Linux User's Guide* has further details on calling C language functions from Fortran.

Intel Fortran Compiler for Linux uses a different name-mangling scheme than the g77 compiler. Intel does not recommend mixing object files created by the Intel Fortran Compiler and the g77 compiler.

Customer Feedback

Intel is committed to providing compilers that deliver the highest Linux-based application performance for applications running on systems that use IA-32, processors with Intel EM64T, or Itanium 2 processors . Intel Premier Support is included with every purchased compiler; see <http://www.intel.com/software/products> for more information.

Intel strongly values customer feedback and is interested in suggestions for improved compatibility with the GNU compilers. If your applications require additional compatibility features, please submit a feature request by creating a customer-support issue through your Intel Premier Support account that explains your request and its impact.

Developers should register for an Intel Premier Support account to obtain technical support and product updates. The product-release notes describe how to register.

Conclusion

Intel C++ Compilers 9.0 for Linux continues to provide outstanding source-language, binary, and command-line compatibility with the GNU C and C++ Compilers. This compatibility gives significant advantages to developers by increasing flexibility while enabling software to run at top speeds on IA-32, processors with Intel EM64T, and Itanium processor-based systems. Intel encourages users to submit feature requests for enhancements in this area.

Appendix A

Supported Linux OS information

The following table lists Linux OS support information for recent releases of the Intel C++ Compiler for Linux. Supported processor architectures are IA-32, Intel® Extended Memory 64 Technology (Intel® EM64T), and Itanium® architecture. Consult the product release notes for the most recent product information.

References

- <http://www.intel.com/software/products>
Provides general information on Intel® software development tools, including the Intel C++ and Fortran Compilers
- <http://developer.intel.com/software/products/opensource/>
Provides documentation, application notes, and source code examples
- <http://developer.intel.com/design/itanium/family>
Contains information on Itanium processor architecture
- <http://developer.intel.com/design/pentium4/>
Supplies Pentium 4 processor information
- <http://www.openmp.org> Contains information on the OpenMP standard
- <http://www.ansi.org> Provides information on the ANSI C and C++ standards
- <http://www.gnu.org> Contains information on the GNU project including GNU GCC and g77 compilers and glibc, the GNU C library
- <http://www.codesourcery.com/cxx-abi/> Describes conventions for object code interfaces between C++ code and implementation-provided system and libraries
- <http://www.intel.com/pressroom/archive/releases/20040803net.htm> Press release from Intel Corporation and Red Flag Software Co., Ltd, announcing that Red Flag is the first company to use the Intel C++ Compiler for Linux to compile a commercial version of its Linux operating system.

Intel® C++ Compiler for Linux* version	Processor Architecture	Supported glibc versions	Supported kernel versions	Supported g++ versions for binary compatibility
9.0	IA-32	2.2.4, 2.2.5, 2.3.2	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x
9.0	Intel® EM64T	2.2.4, 2.2.5, 2.3.2	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x
9.0	Itanium® Architecture	2.2.4, 2.2.5, 2.3.2	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x
8.1	IA-32	2.2.4, 2.2.5, 2.3.2	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x
8.1	Intel EM64T	2.2.4, 2.2.5, 2.3.2	2.4.x, 2.6.x	3.2.x, 3.3.x, 3.4.x
8.1	Itanium Architecture	2.2.4, 2.2.5, 2.3.2	2.4.x	3.2.x, 3.3.x, 3.4.x
8.0	IA-32	2.2.5, 2.2.93, 2.3.2	2.4.x	3.2.x, 3.3.x
8.0	Itanium Architecture	2.2.4, 2.2.5, 2.3.2	2.4.x	3.2.x, 3.3.x



Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052-8119
USA

For product and purchase information visit:
www.intel.com/software/products

Intel, the Intel logo, Pentium, and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.