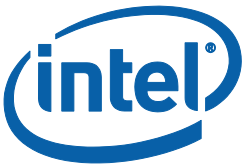

Intel® C++ Compiler v14.0 for Android* Installation Guide and Release Notes

Document number: 327050-001US

22 January 2014

Table of Contents

1 Introduction

This document describes how to install the product, provides a summary of new and changed features and includes notes about features and problems not described in the product documentation.

Intel® C++ Compiler v14.0 for Android* is a standalone product based on the Intel® C++ compiler v14.0 in Intel® C++ Composer XE 2013 SP1 with some limitations.

1.1 Change History

This section highlights important changes in product version 14.0 update 1

- Default installation directory changed on Windows*
- Support for Android NDK r9b & r9c
- KitKat support
- Support multiple Android* NDKs
- Limited support of vs-android plugin tool on Windows
- Other bug fixes

This section highlights important changes in product version 14.0

- Compiler libraries are linked statically by default. Need to use –shared-intel to revert to old behaviour.
- NDK_TOOLCHAIN name changed from '`icc`' to '`x86-icc`'
- The compiler support NDK R9 and cannot work with STLport from NDK 8E
- STLport C++ implementation is now supported by Pointer Checker and Cilk Plus.
- Directory structure changed.
- GUI installer
- Installer does the NDK integration
- Setting ANDROID_SYSROOT and ANDROID_GNU_X86_TOOLCHAIN environment variables are no longer required
- Eclipse help integration
- New standalone product on Windows and OS X* targeting Android*
- 'iccvars' script file no longer available
- Support for Android NDK r9
- Other minor changes.

Changes in the product version 13.0.1.016

- New option '-xatom_sse4.2' enabling optimizations for Silvermont target architecture.
- Support for Android NDK r8e.
- Compatibility and stability fixes for Android* Open Source Project (AOSP) build.
- Performance improvements.

1.2 Product Contents

Intel® C++ Compiler for Android* includes the following components:

- Intel® C++ Compiler v14.0 for building applications that run on X86 architecture systems running the Android* operating system
- On-disk documentation

1.3 System Requirements

For an explanation of architecture names, see http://intel.ly/q9JVjE

- A PC based on an IA-32 or Intel® 64 architecture processor supporting the Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instructions (Intel® Pentium® 4 processor or later, or compatible non-Intel processor)
- For the best experience, a multi-core or multi-processor system is recommended
- 1GB of RAM (2GB recommended)
- 2GB free disk space for all features
- The compiler does require an Android* native build environment to work. One of the following is required:
    - Android* NDK r9, r9b or r9c (32-bit or 64-bit)
    - Android* Open Source Project (AOSP) workspace, for example AOSP workspace for Android 4.3.
- Eclipse* users, please install the following:
    - Android* SDK ADT bundle (32-bit or 64-bit, 20131030 or 20130917 package)
- Supported target devices:
    - Intel® Atom™ processor devices with Android* Jelly Bean
    - Intel® Atom™ processor devices with Android* KitKat

1.3.1 Windows host

- Microsoft Windows 7* (32-bit or 64-bit), Windows 8* or 8.1* (32-bit or 64-bit)
- To use Intel C++ Compiler for Android within Microsoft* Visual Studio
    - Install “vs-android” version 0.96 and all required components
    - Android NDK r9c is required for “vs-android” v0.96
    - Please read Limited support of vs-android plugin tool on Windows section below.

1.3.2 OS X* host

- A 64-bit Intel®-based Apple* Mac* system host
- OS X* version 10.8 or 10.9

1.3.3 Linux host

- The following Linux distributions are supported and tested (other distributions may or may not work -please refer to Technical Support if you have questions):
    - 32-bit Ubuntu* 11.04 or later
    - 64-bit Ubuntu* 12.04
- Linux Developer tool components installed, including gcc, g++ and related tools

- Development for a 32-bit on a 64-bit host may require optional library components (ia32-libs, lib32gcc1, lib32stdc++6, libc6-dev-i386, gcc-multilib) to be installed from your Linux distribution.

Notes

- The default for the Intel® compilers is to build IA-32 architecture applications that require a processor supporting the Intel® SSE2 instructions -for example, the Intel® Pentium® 4 processor. A compiler option is available to generate code that will run on any IA-32 architecture processor.
- Compiling very large source files (several thousands of lines) using advanced optimizations such as -O3 and -ipo may require substantially larger amounts of RAM.
- The above lists of processor model names are not exhaustive -other processor models correctly supporting the same instruction set as those listed are expected to work. Please refer to Technical Support if you have questions regarding a specific processor model
- Some optimization options have restrictions regarding the processor type on which the application is run. Please see the documentation of these options for more information.

1.4 Documentation

Product documentation can be found in the Documentation folder as shown under Installation Folders. Please note that the documentation is generic for Intel® C++ Compiler products and hence contains features not supported by the Intel® C++ Compiler for Android. For example the Visual Studio and Xcode integration is not applicable for the Intel® C++ Compiler for Android. Please refer to the Getting Started document that contains information solely for the Intel® Compiler for Android*.

Optimization Notice

Intel‘s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

1.5 Technical Support

For information about how to find Technical Support, Product Updates, User Forums, FAQ, tips and tricks and other support information, please visit http://www.intel.com/software/products/support/ or contact your Intel representative.

2 Installation

The installation of the product requires a valid license file or a serial number except for an evaluation copy.

You will receive the product as a downloadable file.

2.1 Windows host

Please double-click on the downloaded file (.EXE) to begin installation.
You do not need to uninstall previous versions or updates before installing a newer version – the new version will coexist with the older versions

2.1.1 Installation Folders

The compiler is, by default, installed in the following directory: `%SystemDrive%\Intel\INDE\Intel_compiler\cc_android_14.x.y.zzz`. You can select a different location during installation time. You do not need to uninstall previous versions or updates before installing a newer version – the new version will coexist with the older versions.

2.1.2 Removal / Uninstall

Use the Windows Control Panel "Add or Remove Programs" to remove the product.

2.2 OS X* host

You will need to have administrative or "sudo" privileges to install, change or uninstall the product.
Please double-click the downloaded file and follow the prompts to complete installation.

2.2.1 Installation Folders

The compiler installs, by default, under `/opt/intel/cc_android_14.x.y.zzz` – this is referenced as `<install-dir>` in the remainder of this section. You are able to specify a different location at installation time. You do not need to uninstall previous or updates before installing a newer version – the new version will coexist with the older versions.

2.2.2 Removal / Uninstall

1. Open Terminal and set directory (cd) to the <install-dir>
2. Type the command:

```
$ sudo ./uninstall_cc_android.sh
```

3. Follow the prompts

2.3 Linux host

First unpack it into a writeable directory of your choice using the command:

```
tar -xzvf name-of-downloaded-file
```

Then change the directory (cd) to the directory containing the unpacked files and begin the installation using the command:

```
./install.sh
```

or

```
./install_GUI.sh
```

Follow the prompts to complete installation.

You do not need to uninstall previous versions or updates before installing a newer version – the new version will coexist with the older versions.

2.3.1 Known Installation Issues

If you have enabled the Security-Enhanced Linux (SELinux*) feature of your Linux distribution, you must change the SELINUX mode to permissive before installing the Intel C++ Compiler. Please see the documentation for your Linux distribution for details. After installation is complete, you may reset the SELINUX mode to its previous value.

If you get the following error message during installation:
"Based on availability of RPM packages and coexistence checks, there is nothing to install from this package. Installation will quit."
The most likely reason is that the package is already installed. If you would like to remove the existing package please use the uninstaller. You find the uninstaller in the folder where you installed the product.

2.3.2 Installation Folders

The compiler installs, by default, under /opt/intel/cc_android_xxx – this is referenced as <install-dir> in the remainder of this section. You are able to specify a different location, and can also perform a "non-root" install in the location of your choice.

2.3.3 Removal/Uninstall

Removing (uninstalling) the product should be done by the same user who installed it (root or a non-root user). If sudo was used to install, it must be used to uninstall as well.

1. Open a terminal window and set directory (cd) to the <install-dir>
2. Type the command: `./uninstall.sh` or `./uninstall_GUI.sh`
3. Follow the prompts

.

2.4 Intel® Software Manager

The installation now provides an Intel® Software Manager to provide a simplified delivery mechanism for product updates and provide current license status and news on all installed Intel® software products.

You can also volunteer to provide Intel anonymous usage information about these products to help guide future product design. This option, the Intel® Software Improvement Program, is not enabled by default – you can opt-in during installation or at a later time, and may opt-out at any time. For more information please see http://intel.ly/SoftwareImprovementProgram.

3 New Features in Intel® C++ Compiler for Android

3.1 New Features in update 1

3.1.1 Default installation directory changed on Windows

The default installation directory is changed from in update 1. The Installation section above is using the new installation directory structure.

- Old default: `%SystemDrive %\Intel\cc_android_14.x.y.zzz`
- New default: `%SystemDrive%\Intel\INDE\Intel_compiler\cc_android_14.x.y.zzz`

3.1.2 Supports multiple Android* NDKs

The update 1 release supports multiple Android NDKs including Android NDK r9, r9b or r9c.

The Intel C++ Compiler is integrated into one Android NDK specified during installation just like before. If you would like to integrate the Intel C++ Compiler into additional installations of the Android NDK, the new script tool below coming with Intel C++ Compiler for Android can be used to simplify the steps:

- <icc-install-dir>/toolchains/ndk-integration.sh on Linux or OS X
- <icc-install-dir>\toolchains\ndk-integration.bat on Windows

Please reference to this article Intergrading the Intel(R) C++ Compiler for Android* with multiple Android NDKs for detail instructions.

3.1.3 Limited support of vs-android on Windows

The update 1 release supports “vs-android” version 0.96. The installation of Intel(R) C++ Compiler for Android will install the corresponding platform toolset when it detects the vs-android plugin during installation.

This support makes it possible to use Intel C++ Compiler for Android in Microsoft* Visual Studio*. Please refer to this article Using the Intel(R) C++ Compiler for Android* with Microsoft Visual Studio* for more detail instructions.

4 Intel® C++ Compiler

This section summarizes changes, new features and late-breaking news about the Intel C++ Compiler.

4.1 Compatibility

The IA-32 architecture default for code generation assume that Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instructions are supported by the processor on which the application is run. This is inherited from the classic Intel® C/C++ Compiler XE 14.0 for Linux*. See below for more information.

4.2 New Features

The following features are new or significantly enhanced in this version. For more information on these features, please refer to the documentation.

- Features from C++11 (-std=c++11): refer to the online article C++11 Features Supported by Intel® C++ Compiler
  - Complete (instead of partial) implementation of initializer lists. See N2672 and N3217.
  - Complete implementation of inline namespaces. See N2535.
  - Complete implementation of non-static data member initializers. See N2756.
  - Complete implementation of generalized constant expressions. See N2235.
  - Complete implementation of unrestricted unions. See N2544.
  - Delegating constructors. See N1986.
  - Rvalue references for *this. See N2439.
  - Raw string literals. See N2442.
  - Conversions of lambdas to function pointers.
  - Implicit move constructors and assignment operators. See N3053.
  - __bases and __direct_bases type traits.
  - The context-sensitive keyword "final" can now be used on a class definition, and "final" and "override" can be used on member function declarations. See N2928, N3206, and N3272.
  - Complete implementation of the "noexcept" specifier and operator. See N3050. Includes the late instantiation of noexcept per core issue 1330.
- Intel® Cilk™ Plus changes
- DWARF V4 support
- `__INTEL_COMPILER_UPDATE` predefined macro
- Pointer type alignment qualifiers
- Variable definition attributes to avoid false sharing
- `-mtune` performance tuning option

4.2.1 Intel® Cilk™ Plus changes

New features for Intel® Cilk™ Plus in Intel C++ Compiler v14.0 update 1 for Android:

- A new Intel® Cilk™ Plus STL vector reducer: a `reducer_vector` class is now provided. The header file "`cilk/reducer_vector.h`" will need to be included. The reducer type is

`cilk::reducer< cilk::op_vector<type> >`. See the header file comments for further specifics.

- The `this` pointer is now allowed in the Intel® Cilk™ Plus SIMD-enabled function uniform clause (i.e. `__declspec(vector(uniform(this)))`)

New features for Intel® Cilk™ Plus in Intel C++ Compiler v14.0 for Android:

- Elemental function implementation has changed to be more compatible with other vector function implementations in gcc and OpenMP*. This breaks binary compatibility with previous Intel® C++ Compiler versions (13.1 and earlier). You should either rebuild all codes using elemental functions with the version 14.0 compiler, or use the `-vecabi=legacy` compiler option to use the previous implementation.
- New multiply reducer defined in `cilk/reducer_opmul.h`
- Three new array notation reduction intrinsics have been added to support bitwise reduction operations:
  - `__sec_reduce_and`
  - `__sec_reduce_or`
  - `__sec_reduce_xor`

4.2.2 New attribute for pointers and pointer types to specify assumed data alignment

`__declspec(align_value(N))` and `__attribute__((align_value(N)))` have been added to indicate to the compiler it can assume the specified alignment "N" when using the attributed pointer type.
For example:

```
typedef float float_a16
__attribute__((align_value (16)));

void foo(float_a16 *restrict dest, float_a16 *restrict src){
```

tells the compiler that the src and dest arguments should be aligned by the user on 16- byte boundaries.

4.2.3 New __INTEL_COMPILER_UPDATE predefined macro

A new `__INTEL_COMPILER_UPDATE` predefined macro can now be used to obtain the minor update number for the Intel® Compiler being used. For example, for a compiler version 14.0.2, the macro would preprocess to "2".

4.3 New Compiler Options

For details on these and all compiler options, see the Compiler Options section of the on-disk documentation.

4.3.1 New options

- `-opt-assume-safe-padding`
- `-opt-prefetch-distance:n1[,n2]`
- `-opt-streaming-cache-evict[:n]`
- `-opt-threads-per-core:n`
- `-xATOM_SSE4.2`
- `-xATOM_SSSE3`
- `-vecabi=<arg>`

- `-gdwarf-4`
- `-standalone`
- `-mtune=<arch>`
- `-W[no-]pch-messages`
- `-mtune=<arch>`

Please note that the Windows hosted compiler will accept options in the format described above [i.e. 'in Linux format']. For a list of deprecated compiler options, see the Compiler Options section of the documentation.

4.3.2 –mtune added

-mtune=<arch> can now be used to specify the compiler "tuning" for a specific architecture, similar to how the equivalent gcc* option behaves.

4.3.3 –gcc-version is deprecated

-gcc-version functionality has been superseded by –gcc-name. –gcc-version has therefore been deprecated and may be removed from a future release.

4.4 Other Changes

4.4.1 Full support for latest Atom™ Processor instruction set

The compiler does support the instruction set of the latest Atom™ Processor, if the option -xatom_sse4.2 is set.

4.4.2 New directory structure after installing Intel® C++ Compiler

After installing the Intel® C++ Compiler v14.0 for Android you find the following directory structure for Linux and OS X hosts:

../cc_android_xxx
- ./bin
- ./bin/ia32
- ./compiler
- ./compiler/lib
- ./compiler/lib/ia32
- ./compiler/lib/ia32/locale
- ./compiler/lib/ia32/locale/en_US
- ./compiler/lib/ia32/gnustl
- ./compiler/lib/ia32/stlport
- ./compiler/include
- ./compiler/include/cilk
- ./compiler/include/ia32
- ./Documentation

For Windows hosts a minor difference in the directory structure can be found:

../cc_android_xxx
./bin
./bin/ia32
./bin/ia32/1033
./compiler
./compiler/lib
./compiler/lib/ia32
./compiler/lib/ia32/gnustl
./compiler/lib/ia32/stlport
./compiler/include
./compiler/include/cilk
./compiler/include/ia32
./Documentation

4.4.3 Establishing the Compiler Environment

The compilervars.[c]sh script is used to establish the compiler environment on Linux and OS X as:

$ “source <install-dir>/bin/compilervars.[c]sh”

On Windows use following command to set up the compiler environment:

- `"<install-dir>\bin\compilervars.bat"`

4.4.4 __attribute__((always_inline)) now requires inline keyword to enable inlining

In previous Intel compiler versions, a routine declared with the "always_inline" attribute would always be inlined. The compiler now requires that the routine also be inline (either explicitly declared that way using the "inline" keyword or implicitly inline because it is a member function whose definition appears inside the class) in order for the routine to be inlined. The compiler will now match gcc behavior and also give a warning for this, i.e.:

```
// t.cpp
__attribute__((always_inline)) int foo2(int x) // need to add
"inline" keyword also
{
return x;
}

icpc -c t.cpp
t.cpp(2): warning #3414: the "always_inline" attribute is ignored on
non-inline functions
__attribute__((always_inline)) int foo2(int x)
      ^
```

4.5 How to use Intel® C++ Compiler for building Android* Applications

Please also see the 'Getting Started Guide' in <install_dir>/Documentation for additional information, e.g. with default installation directory:

- "/opt/intel/cc_android_xxx/Documentation/en_US/m_cc_getting_started.htm" on OS X
- "/opt/intel/cc_android_xxx/Documentation/en_US/l_cc_getting_started.htm" on Linux
- "c:\intel\cc-android-xxx\Documentation\en_US\w_cc_getting_started.htm" on Windows

Please reference the article Using Advanced Intel® C++ Compiler Features for Android* Applications for up to date information on using some of the advanced Intel® C++ Compiler features for Android* applications.

4.6 Other Notes

4.6.1 Instruction Set Default require Intel® Streaming SIMD Extensions 2 (Intel® SSE2)

This is inherited from the classic Intel® C++ Compiler XE 14.0 for Linux*.

When compiling for the IA-32 architecture, -msse2 (formerly -xW) is the default. Programs built with –msse2 in effect require that they be run on a processor that supports the Intel® Streaming SIMD Extensions 2 (Intel® SSE2), such as the Intel® Pentium® 4 processor and some non-Intel processors. If the program is run on an unsupported processor, an invalid instruction fault may occur. Note that this may change floating point results since the Intel® SSE instructions will be used instead of the x87 instructions and therefore computations will be done in the declared precision rather than sometimes a higher precision.

To specify generic IA-32, specify –mia32.

4.6.2 Features not carried forward from Intel® C++ Compiler XE 14.0

The following features in the Intel® C++ Compiler XE 14.0 are not supported in current release:

- OpenMP* support.
- Eclipse integration except for help information

4.7 Known Issues

4.7.1 Guided Auto-Parallel Known Issues

Guided Auto Parallel (GAP) analysis for single file, function name or specific range of source code does not work when Whole Program Interprocedural Optimization (-ipo) is enabled.

4.7.2 Intel® Cilk™ Plus Runtime Support Limitations

The Intel® Cilk™ Plus runtime support for Android has the following limitations:

- Inter-operability with OpenMP, Intel® Threading Building Block (Intel® TBB) is not supported
- Inter-operability with Intel® Vtune™ Amplifier is not supported

- C++ exceptions cannot be thrown across a cilk_spawn or cilk_for boundary

4.7.3 Intel® Cilk™ Plus Runtime and standard C++ library

The Intel® Cilk™ Plus Runtime library depends on the stack unwinding for exception handling provided by a standard C++ library. There are several standard C++ library incompatible implementations on Android differing by the set of supported features.

Intel® C++ Compiler for Android provides two runtime libraries:

1. one compatible with stlport_shared C++ implementation;
2. one compatible with gnustl_shared and gnustl_static C++ implementation.

Other C++ implementations from NDK are not supported with Intel ® Cilk™ Plus technology. In the case application does not depend on C++ runtime, it is safe to use gnustl_static C++ implementation.

5 Disclaimer and Legal Information