Intel® C++ Compiler for Windows* Compatibility with Versions of Microsoft* Visual C++*
# Table of Contents

Introduction .................................................................................................................................................................................. 3

Overview ......................................................................................................................................................................................... 3

General Compatibility .................................................................................................................................................................... 4

Compatibility Options ..................................................................................................................................................................... 4

Microsoft Visual C++ Pragmas Support ......................................................................................................................................... 4

Differences in Treatment of Inline Assembly Labels ................................................................................................................ 5

Differences with Macro Expansions .............................................................................................................................................. 5

Differences in Precompiled Header (PCH) Support .................................................................................................................. 5

Differences in the enum type ......................................................................................................................................................... 5

Microsoft Visual C++ 6.0 Compatibility ..................................................................................................................................... 6

Integration with Visual C++ 6.0 IDE ............................................................................................................................................... 6

Visual C++ 6.0 Processor Pack Compatibility ........................................................................................................................ 6

Intel IA-64 -based Processor Targeting ...................................................................................................................................... 6

Unsupported Compiler Options ..................................................................................................................................................... 6

Unsupported Pragmas ....................................................................................................................................................................... 6


Integration with Visual C++ .NET 2003 and Visual Studio 2005 IDEs .......................................................................................... 7

New Features of the IDE Integration ............................................................................................................................................ 8

Automation Interfaces ....................................................................................................................................................................... 8

Unsupported Visual C++ Project Types ......................................................................................................................................... 8

Unsupported Major Language Features ........................................................................................................................................ 9

Unsupported Preprocessor Features ........................................................................................................................................... 9

Whole Program Optimization and Interprocedural Optimization ................................................................................................. 10

Mixing Unmanaged Code Compiled with the Intel C++ Compiler with Managed Code ............................................................... 11

The Intel*-64-based Processor and IA-64-based Processor Targeting ....................................................................................... 11

Compatibility in OpenMP* Support ............................................................................................................................................ 12

References ....................................................................................................................................................................................... 13
Introduction

The Intel® C++ Compiler 10.0 for Windows® supports the Microsoft* Visual C++* extensions to the C and C++ languages. This document describes the important differences of the Intel C++ Compiler 10.0 and Visual C++ extensions.

Overview

The Intel C++ Compiler is source- and binary- (native code) compatible with the Microsoft* Visual C++* Compiler. It is also well integrated into the following Microsoft development environments: Microsoft Visual C++* 6.0, Microsoft Visual C++ .NET 2003, and Microsoft Visual C++ 2005. The Intel C++ Compiler generates optimized code with the advanced features of Intel* IA-32 processors, Intel® 64 (previously named EM64T), and the Intel IA-64 (previously named Itanium* ) processors.

Because of the Intel C++ Compiler's source- and binary-compatibility with the Microsoft Visual C++ Compiler, it enables you to harness the performance-enhancement features of the Intel C++ Compiler by selectively rebuilding only the parts of your application which you deem to be performance sensitive with the Intel Compiler. If necessary, you can mix and match object files and libraries built with either of the two compilers.

The Intel C++ Compiler for Windows conforms to the following standards:

- OpenMP* 2.5 API specification

It is important to note that the 10.0 version of the compiler is no longer supporting Microsoft Visual C++ .NET (2002).

One new feature of the Intel C++ Compiler 10.0 is integrated support for the Static Verifier. The related options to invoke and manipulate the functionality are:

- /Qdiag-enable:sv{1|2|3}.
- /Qdiag-enable:sv-include.

When the Static Verifier is invoked, the Intel C++ Compiler will perform the necessary analysis and flag potential errors or questionable areas in the source code. The Static Verifier can detect many difficult source code issues at compile-time, such as incorrect usage/modification of an object, incorrect memory usage (uninitialized variables, boundaries violations, or memory leaks), inconsistent object declarations, inheritance issues, C++/STL issues (operator new/delete, overloaded operator issues, uncaught exceptions), bad parameters to intrinsics, dead code and redundant executions etc. It can analyze C/C++ code and OpenMP code alike and is a great tool for debugging your applications. Detailed information can be found in the User’s Guide.

General Compatibility

Compatibility Options
The following Intel Compiler options provide compatibility with different versions of Microsoft Visual C++:

- /Qvc6 Microsoft Visual C++ 6.0.

The Intel C++ Compiler has provided an option, /Qms[0,1,2], to simulate some of the particular behavioral characteristics of the Visual C++ compiler. In a few cases, the Microsoft Compiler will be able to compile source code without any errors while the Intel Compiler will generate errors for the same source code. You can control the level of compatibility with the /Qms[0,1,2] option:

- /Qms0 instructs the compiler to disable Microsoft compatibility bugs.
- /Qms1 instructs the compiler to enable most Microsoft compatibility bugs (default).
- /Qms2 instructs the compiler to implement full Microsoft compatibility.

The following macro has been predefined in the Intel C++ Compiler to specify the level of Microsoft compatibility provided:

- __INTEL_MS_COMPAT_LEVEL__ - the value of this macro is [0,1,2] that is controlled by the -Qms option.

Microsoft Visual C++ Pragmas Support
The Intel C++ Compiler supports Microsoft Visual C++ pragmas but has the following limitations:

- #pragma optimize("", [on|off]) - The Intel C++ Compiler only supports the enabling and disabling of the specific optimizations specified by the compiler options. It does not accept a list of optimizations like Visual C++ does.
- The Intel Compiler accepts the following pragmas without error, but they will have no effect:

  - component
  - inline_recursion
  - function
  - intrinsic
  - include_alias
  - setlocale
  - inline_depth
Differences in Treatment of Inline Assembly Labels
The Intel C++ Compiler treats the inline assembly labels in a case-sensitive manner. In contrast, the Microsoft Visual C++ Compiler treats them in a case-insensitive manner.

Differences with Macro Expansions
The two compilers act differently when, in the source code, a macro is passed as a parameter to another macro using the token concatenation operator to generate new code.

The Intel C++ Compiler does not expand the macro used as a parameter before concatenation. Nor does it expand macros passed to #include directives.

The Microsoft Visual C++ Compiler performs an extra preprocessing scan to expand those macros.

Differences in Precompiled Header (PCH) Support
The precompiled header (PCH) information generated by the Intel C++ Compiler is not compatible with the PCH information generated by the Microsoft Visual C++ compiler. The specific incompatibilities are as follows:

- The Intel Compiler does not recognize the PCH file generated by the Microsoft compiler. If no suitable or recognized PCH file exists, compilation proceeds without use of PCH files.
- The Microsoft compiler aborts with an error message when it tries to use a PCH file generated by the Intel compiler.
- The Intel C++ Compiler does not support PCH generation and use in the same translation unit.
- The Intel C++ Compiler does not generate PCH information beyond the point where a declaration is seen in the primary translation unit. When the /Yu option is specified, the Microsoft Visual C++ compiler ignores all text, including declarations preceding the #include statement of the specified file.
- PCH files coexistence: the /Qpchi option causes the Intel C++ Compiler to name its PCH files with a .pchi filename suffix and reduces build time. The /Qpchi option is on by default; use /Qpchi- to turn it off.

Differences in the Enum Type
Microsoft Visual C++ always considers enum bit fields to be signed, even if not all values of the enum type can be represented by the bit field. On the other hand, the Intel C++ Compiler considers an enum bit field to be unsigned, unless the enum type has at least one enum constant with a negative value.
Microsoft* Visual C++* 6.0 Compatibility

The Intel C++ Compiler is fully source- and binary-compatible with the Microsoft Visual C++ 6.0 Compiler when "/Qvc6" is specified. You can build either the full set of project files or part of the set of project files with the Intel C++ Compiler.

Integration with Visual C++* 6.0 IDE

The Intel C++ Compiler is integrated into Microsoft Visual C++ 6.0 IDE through the Intel® C++ Compiler Selection Tool that is available from the [Tools] > [Intel(R) C++ Compiler Selection Tool] menu. Binaries built with the Intel C++ Compiler can be debugged from within the Microsoft Visual C++ 6.0 IDE. From within the Microsoft Visual C++ 6.0 IDE, programs can be built by the Intel C++ Compiler to target either IA-32-based systems or IA-64(Itanium®)-based systems.

Visual C++* 6.0 Processor Pack Compatibility

If your program uses the Single Instruction Multiple Data (SIMD) C++ data types, they may be affected by changes in the binary conventions used for compatibility with the Microsoft Visual C++ 6.0 Processor Pack. These data types include:

- __m64
- __m128d
- __m128
- __m128i

The /Qmspp [-] option, available in the Intel C++ Compiler for IA-32 system, is ON by default. It enables binary compatibility with Microsoft Visual C++ 6.0 Processor Pack.

Intel IA-64 -based Processor Targeting

The Intel C++ Compiler has been integrated into the Microsoft Visual Studio* 6.0 IDE to facilitate the development of applications for IA-64-based systems using the Intel C++ Compiler Selection Tool. The latest Platform SDK is required for developing the IA-64-based application.

You can also develop IA-64-based applications from the command window with the Intel C++ Compiler.

Unsupported Compiler Options

The Intel C++ Compiler supports most of the options of the Microsoft Visual C++ 6.0 Compiler. However, it does not support the options listed in the table below. Most of the unsupported options are useful during development time and are typically not required to build a working application.

Unsupported Pragmas

Refer to the earlier discussion in this paper, Microsoft Visual C++* pragmas support.

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Fd</td>
<td>Name of the PDB file used for debug information for specified source files.</td>
</tr>
<tr>
<td>/Gi</td>
<td>Enables incremental compilation.</td>
</tr>
<tr>
<td>/ZI</td>
<td>Edits and continues debugging (similar effect as the /Zi option).</td>
</tr>
<tr>
<td>/Gm</td>
<td>Enables minimal rebuild.</td>
</tr>
<tr>
<td>/Yd</td>
<td>A Microsoft precompiled header-specific option that puts debug information in every object.</td>
</tr>
<tr>
<td>/2mn</td>
<td>Controls maximum memory allocated by the compiler.</td>
</tr>
</tbody>
</table>

Table 1. Unsupported Compiler Options of the Visual C++ 6.0
The Intel C++ Compiler is fully source- and binary- (native code only) compatible with the Microsoft Visual C++ .NET 2003 compiler and the Visual C++ 2005 compiler when the option "/Qvc7.1" or "/Qvc8" is specified. Binaries built with the Intel C++ Compiler can be debugged from within the Microsoft Visual C++ .NET 2003 IDE or Visual Studio 2005 IDE.

With regard to security checks, /GS is supported by the Intel C++ Compiler, but it is disabled by default; in contrast, in Visual C++ 2005, it is enabled by default. Please specify /GS to enable the security checks with the Intel C++ Compiler. The Intel C++ Compiler for Windows supports the safe exception handler feature with option /QsafeSEH for 32-bit binary and is on by default.

Manifest is a Visual Studio 2005 feature that describes run-time dependencies of an application. The Intel C++ Compiler supports the generation of manifest files, which are typically placed alongside the EXE or DLL, with a file type of .manifest. Manifest files can also be embedded in the EXE or DLL, but this capability is supported only when incremental linking is disabled.

Integration with Visual C++ .NET 2003 and Visual Studio 2005 IDEs

The Intel C++ Compiler for Windows is integrated into Microsoft Visual C++ .NET 2003 and Visual Studio 2005 IDEs through the Visual Studio Industry Partner (VSIP) interfaces. A new toolbar from the Intel C++ Compiler is added to both IDEs after installing the Intel C++ Compiler for Windows.

Microsoft Visual C++ .NET 2003 and Visual Studio 2005 maintain a .vcproj file for the Visual C++ project, and the Intel C++ project system maintains a corresponding file, .icproj, that contains the configuration data specific to the Intel C++ Compiler. In order to use the Intel C++ Compiler, you need to convert your project or solution to use the Intel C++ project system first. After converting, a new .icproj file will be created for every .icproj project.

There are two ways to convert a Microsoft Visual C++ .NET 2003 or Visual C++ 2005 project or solution to use the Intel C++ project system:

- From the Intel C++ Compiler's command window, use "ICProjConvert100.exe".
- From the Microsoft Visual C++ .NET 2003 or 2005 IDE, you can convert the project by following the steps below:
  1. Right-click on a project, multiple projects or the solution and select the menu item "Convert to use Intel C++ Project System".
  2. Select menu item [Project] -> [Convert to use Intel C++ Project System].
  3. Click the Intel C++ Compiler's toolbar icon.

When you develop a project among your development group, but not everyone in your group has the Intel C++ Compiler installed, you need to share the .vcproj file among the group. The previous conversion tool "ICProjConvert90.exe" requires the "/sharevcproj" option so that the .vcproj file will not be modified when converting a project to the Intel C++ Project System. The new version of this tool "ICProjConvert100.exe" doesn't require this option any more, since the behavior that was enabled by the option is now the default. A new option, "/nonsharedvcproj", has been added if you do not want the .vcproj file to be shared.
In previous versions of the Intel IDE integration, you could control this functionality from the Intel Tool Options page at [Tool] > [Options] > [Intel® C++] > [General page]. With 10.0, this choice is only available within Visual C++ .NET 2003, not Visual C++ 2005.

The following capabilities are provided by the integration:

- Converting one or more Visual C++ projects in a solution to use the Intel C++ Project System.
- Selecting either the Intel C++ Compiler or the Microsoft Visual C++ Compiler to build each configuration of a project or individual files within a project.
- Optimizing your application using compilation options specific to the Intel C++ Compiler.
- Selection of the version of the Intel C++ Compiler to use when multiple versions are installed on your system.

New Features of the IDE Integration

The following features have been added to the Intel C++ Compiler 10.0 for Windows:

- Improved support for the Remote Debugging feature, to match the exact behavior of Visual C++.
- Support for the automation interfaces of the Intel C++ Project System (see the next section for details).
- A new toolbar has been added.

The functionality associated with each icon on the toolbar is "Intel C++ Quick Start", "Convert to use Intel C++ Project System", and "Convert to use Visual C++ Project System".

Automation Interfaces

The Intel C++ Integrations for Visual C++ .NET 2003 and Visual Studio 2005 provide a set of automation interfaces. This is a new feature of the Intel C++ Compiler for Windows 10.0. Experienced developers can use the following automation interfaces to obtain information about how to manipulate Intel C++ projects:

- ICProjectEngine, ICICollection, ICProject, ICPlatform, ICConfiguration, ICFile, IFiler, ICFileConfiguration, ICDDebugSettings.
- OppCompilerTool, ICLinkerTool, ICLibrarianTool, ICRsourceCompilerTool, IMidlTool, ICCustomBuildTool, ICPreBuildEventTool, ICPreLinkEventTool, ICPBuildEventTool, ICBscMakeTool.
- IntelSettings, IntelCompiler, ProjectConversions.

Please refer to the User’s Guide for a full description and examples on how to use these interfaces.

Unsupported Visual C++ Project Types

The Intel C++ Compiler only supports native C++ project types provided by Visual C++ .NET 2003 and Visual Studio 2005. The project types with .NET attributes such as the ones below cannot be converted to an Intel C++ project:

- Empty Project (.NET)
- Class Library (.NET)
- Console Application (.NET)
- Windows Control Library (.NET)
- Windows Forms Application (.NET)
- Windows Service (.NET)

The Intel C++ Compiler for Windows does not support all the features of Microsoft Visual C++ .NET 2003 and Visual C++ 2005. The details are discussed below.
Unsupported Major Language Features

- Attributed code
- Managed extensions for C++ (new pragmas, keywords, and command-line options)
- Event handling (new keywords)
  - __abstract keyword
  - __box keyword
  - __delegate keyword
  - __gc keyword
  - __identifier keyword
  - __nogc keyword
  - __pin keyword
  - __property keyword
  - __sealed keyword
  - __try_cast keyword
  - __w64 keyword

Unsupported Preprocessor Features

- #import directive changes for attributed code
- #using directive
- managed, unmanaged pragmas
  - __MANAGED macro
  - runtime_checks pragma

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/AI&lt;dir&gt;</td>
<td>Adds to assembly search path.</td>
</tr>
<tr>
<td>/clr</td>
<td>Compiles for the common language runtime (managed C++).</td>
</tr>
<tr>
<td>/FU&lt;file&gt;</td>
<td>Forces use of assembly/module.</td>
</tr>
<tr>
<td>/Fx</td>
<td>Merges injected code to file.</td>
</tr>
<tr>
<td>/MPn</td>
<td>Utilizes n processors for compilation.</td>
</tr>
<tr>
<td>/openmp</td>
<td>Use /Qopenmp instead.</td>
</tr>
<tr>
<td>/w&lt;n&gt;</td>
<td>Sets warning level 1-4 for n.</td>
</tr>
<tr>
<td>/wo&lt;n&gt;</td>
<td>Issues warning n once.</td>
</tr>
<tr>
<td>/zm&lt;n&gt;</td>
<td>Maximum memory allocation (percentage of the default).</td>
</tr>
<tr>
<td>/homeparams</td>
<td>Force parameters passed in registers to be written to the stack.</td>
</tr>
<tr>
<td>/doc[fle]</td>
<td>Process XML documentation comments and optionally name the .xdc file.</td>
</tr>
<tr>
<td>/favor: [blend]A MD64</td>
<td>Use /QaxP, /QxW, and /QxP instead. You can find more up to date information at Intel® Compilers Processor-Specific Optimization Options.</td>
</tr>
</tbody>
</table>

Table 2. Unsupported Compiler Options of Visual C++ .NET 2003 and Visual C++ 2005
Whole Program Optimization and Interprocedural Optimization


In order to get optimal performance, we do not recommend mixing and matching object files compiled by different compilers with “Whole Program Optimization” option. Cross-file optimizations cannot be performed amongst files compiled with the Intel compiler and those compiled by the Microsoft compiler. So, all the optimizations of “Whole Program Optimization” that would have been performed if all of the files were compiled with the Intel compiler will not be performed, which could potentially have a dramatically negative impact on performance.

If you do need to mix and match object files, you should use the tools (ie icl, xilink, or xilib) provided with the Intel C++ Compiler. These tools will emit the following warning in this case and continue linking with the Whole Program Optimization disabled:

IPO: WARNING: Whole program optimization will be disabled;
"test.obj" appears to be a Microsoft(R) /GL compiled object file.

Recompile with the Intel(R) compiler using the /GL option for whole program optimization.

If you use the link tools (link.exe, lib.exe) provided with Visual C++, you’ll get the following warning and errors:

  t2.obj : warning LNK4229: invalid directive
            '/Intel(R) compiler non-linkable ILobject file' encountered; ignored
  xxx.obj : error LNK2001: unresolved external
            symbol "int __cdecl foo(void)" (?foo@@YAHXZ)
  xxx.exe : fatal error LNK1120: 1 unresolved
            externals

If you build the project within the Visual Studio IDE, the Intel C++ Compiler integration will cause the correct linking tools to be used.

Visual Studio 2005 provides the following refined options under the “Whole Program Optimization” option:

• Use Link Time Code Generation
• Profile Guided Optimization – instrument
• Profile Guided Optimization – update
• Profile Guided Optimization – optimize

With this option only, if you have some files compiled with the Intel C++ Compiler, you’ll get the following link error:

  LINK : fatal error LNK1269: inconsistent
  file 'ipo_xxxxobj.obj' specified with
  /LTCG:PGOPTIMIZE but not with
  /LTCG:PGINSTRUMENT

The first three options will not cause any link errors. But, the “Profile Guided Optimization –instrument/update” options will not have any effect on the files compiled with the Intel C++ Compiler.
Mixing Unmanaged Code Compiled with the Intel C++ Compiler with Managed Code

If you use the managed extensions to the C++ language in Microsoft Visual C++ .NET, you can use the Intel C++ Compiler for your non-managed code for better application performance. Make sure managed keywords do not appear in your non-managed code.


Note: This URL is available at the Microsoft MSDN* Web site. If it changes, please search for the article.


The Intel*-64-based Processor and IA-64-based Processor Targeting

The Intel C++ Compiler for Windows supports targeting for Intel 64-based and IA-64-based systems from within the Visual Studio 2005 IDE, but you can develop Intel 64-based applications and IA-64-based applications from the command window as well.

The Intel C++ Compiler for Windows does not support targeting for Intel 64-based and IA-64-based systems from within the Visual C++ .NET 2003 IDE.

Note about the Intel C++ Compiler for Intel*-64

To build Intel 64-based applications from within Visual Studio 2005 IDE, you need to install the Visual Studio 2005 Standard edition or above, and the “X64 Compiler and Tools” as well. Also, you should use the “/favor:EM64T” option for best performance on Intel 64-based systems when using Visual C++ 2005, and use “/QxP” when using the Intel C++ Compiler.

The following special features are offered by the Intel C++ Compiler for Intel 64:

1. support for inline asms.
2. support for MMX/x87 instructions through inline asms.
3. support for MMX™ instructions through intrinsics.

Note about the Intel C++ Compiler for IA-64

To build IA-64-based applications within the Visual Studio 2005 IDE, you need to install the Visual Studio 2005 Team System edition of Visual Studio, and you should have the “Itanium Compiler and Tools” component installed.
Compatibility in OpenMP* Support

Visual C++ 2005 now supports the OpenMP* 2.0 API specification. The Intel C++ Compiler for Windows has been supporting OpenMP since version 7.x. The Intel C++ Compiler now supports the OpenMP 2.5 API specification.

The following list briefly summarizes OpenMP support in the Intel C++ Compiler for Windows:

1. Support for the OpenMP 2.5 API specification.
2. Support for the taskqueuing extension.

**Note:** The taskqueuing extension implemented by the Intel C++ compiler extends OpenMP to parallelize a broader range of applications. It allows the user to exploit irregular parallelism with units of work that are not pre-computed at the start of the worksharing construct. Unlike `single`, `for`, and `sections` constructs, all work units are known at the time the construct begins execution.

The taskqueuing pragmas are `taskq` and `task`. The `taskq` pragma specifies the environment within which the enclosed units of work (tasks) are to be executed. From among all the threads that encounter a `taskq` pragma, one is chosen to execute it initially. The `task` pragma specifies a unit of work, potentially executed by a different thread.

3. Implemented extension functions in the OpenMP run-time library and implemented extension environment variables.
4. Support for debugging OpenMP code.
5. Support for enhanced debugging by providing an OpenMP stub library for debugging purposes that links in the OpenMP stub APIs but still runs sequentially; this is supported through `/Qopenmp_stub` option.
6. Support for the Intel® Thread Checker and Intel® Thread Profiler via `/Qtcheck` and `/Qopenmp_profile`.

Because the OpenMP specification does not define the interoperability of multiple implementations, the implementations of the OpenMP by Visual C++ 2005 and Intel C++ Compiler for Windows are not interoperable. To avoid possible linking and runtime problems, please follow the rules below:

1. Avoid using multiple copies of the OpenMP runtime library from different compilers. Because each OpenMP runtime library assumes it has all the resources of the system, a problem with the oversubscription of threads to processors may result.
2. Compile all the OpenMP sources with one compiler, if possible, or at least the parallel region and entire call tree under it within the source. The routines exposed to users should not have orphaned OpenMP constructs.
3. Use the dynamic library for OpenMP.

References

You can find useful information about compiler compatibility in the following documents, each of which is available from the Intel C++ Compiler 10.0 for Windows web site:

- Intel® C++ Compiler 10.0 for Windows Release Notes
- Intel® C++ Compiler 10.0 for Windows User’s Guide
- Optimizing Applications with the Intel® C++ and Fortran Compilers for Windows and Linux