WHITE PAPER

Intel® C++ Compiler 8.1 for Windows*

Getting Started Guide

Intel® software development products

Table of Contents

Overview

This document explains how to install the Intel® C++ Compiler for Windows*; build "Hello World" project in the Visual C++* 6.0, Visual C++ .NET 2002 and Visual C++ .NET 2003 environments for IA-32 and Itanium®-based systems; and how to get started optimizing your applications with the Intel® compilers.

The Intel C++ Compiler 8.1 for Windows consists of the following:

- Intel C++ Compiler for IA–32-based applications: **icl**
  This compiler can be used from a command window or from within the Visual C++ (6.0 or .NET) IDE.
- Intel C++ Compiler for developing Itanium-based applications on IA-32 systems: **icl**
  This compiler can be used from a command window or from within the Visual C++ 6.0 IDE.
- Intel C++ Compiler for Itanium-based applications on Itanium systems: **icl**
  This compiler can be used from a command window only.
- Assembler for IA-32-based systems to produce Itanium-based applications: **ias**
- Assembler for Itanium-based systems to produce Itanium-based applications: **ias**
- Intel® Debugger: **idb**
  The Intel Debugger **idb** on IA-32 systems has a Graphic User Interface, but the Intel Debugger **idb** on Itanium-based systems doesn't have a GUI.
- Intel® License Manager for FLEXlm*
- Utilities:
  - Code-Coverage tool that may greatly help developers improve the development efficiency: **codecov.exe**, located at the same directory as **icl.exe**, default to **`<program files>\intel\cpp\compiler80\bin`** directory.
  - Test-Prioritization tool that may help the developers find the most effective test suites for the application: **tselect.exe**, located at the same directory as **icl.exe**, default to **`<program files>\intel\cpp\compiler80\bin`** directory.
  - Compiler selection tool that integrates Intel C++ Compiler 8.1 with Microsoft Visual C++ 6.0 IDE
  - Compiler integration utilities that integrate Intel C++ Compiler 8.1 with Microsoft Visual C++ .NET IDE 2002 and 2003
  - Project converter utility that converts a Visual C++ .NET project to an Intel C++ compiler project or back: **icProjConvert80.exe**, located at **`<common files>\intel\shared files\ia32\bin`** directory.
  - Utility to support non-administrative users at the start menu **`Programs\Intel® Software Development Tools\Intel® C++ Compiler 8.1\Update User's Registry`**.
  - The Makefile utility that provides users with the ability to switch between the Intel C++ Compiler 8.1 and the Microsoft Visual C++ 6.0 Compiler without requiring changes to their makefiles. The Makefile utility is available from Custom Installation Type only.
  - The **icpi** Utility to isolate compile/link time errors. It is located at **`<installation _ directory>\Compiler80\ia32\bin\icpi.exe`** or at **`<installation _ directory>\Compiler80\Itanium\bin\icpi.exe`**
- Product documentation including:
  - *Intel® C++ Compiler Release Notes*
  - *Intel® C++ Compiler User's Guide*
  - *Intel® Itanium® Assembler User's Guide*
  - *Intel® Itanium® Architecture Assembly Language Reference Guide*
  - *Enhancing Performance with Intel® Compilers* (training tutorial)
  - *Intel® Debugger Manual*
  - *Using the Intel® License Manager for FLEXlm**

System Requirements

For developing applications on IA-32 systems

Refer to the latest *Intel C++ Compiler Release Notes* for details.

For developing Itanium®-based applications on Itanium-based systems

Refer to the latest *Intel C++ Compiler Release Notes* for details.

Installation Notes

The Intel C++ Compiler 8.1 uses the Windows Installer. This provides additional options for customization, update or repair of the installation, as well as providing a single option for uninstalling all components.

The Intel C++ Compiler uses Macrovision's FLEXlm electronic licensing technology. License management should be transparent. A valid license is required for installing and using the Intel C++ Compiler. Please follow the installation steps below to install the FLEXlm license file.

Notes:

- If you are using another version of the Intel C++ Compiler 8.1, we recommend you uninstall it prior to installing this product.
- Before installing the compilers, read the software requirements. Microsoft Platform SDK* should be installed if you are developing Itanium-based applications.

Installing the Intel® C++ Compiler

1. Check the hardware and software requirements. (See above for detail).
2. Install the license.
   The installation program of Intel C++ Compiler checks for a valid license before installing any component. If you have downloaded the compiler from Intel® Premier Support, the license key you received with your Intel C++ Compiler 6.0 or 7.x for Windows will work with the Intel C++ Compiler 8.1 for Windows unless your support services have expired.

Here is how to set up the license file before installation.

- If you have an electronically downloaded version of the Intel C++ Compiler 8.1, the license is sent to you via e-mail. Follow the instructions in the e-mail to install the license file.
- If you have a CD version of the Intel C++ Compiler 8.1, a valid license is included on the CD without support services. The installation program can automatically locate and install the corresponding license for you, to **`c:\program files\Common Files\Intel\Licenses`** on both IA-32 and Itanium-based systems.

  ### Notes for CD-ROM users:

  Do the following to obtain access to technical support and be able to download and execute product updates:

  a. Register: First, locate the serial number found on the inside flap of the product box. Then, visit the Web site http://www.intel.com/software/products/registrationcenter/ and follow the instructions.

  b. Install the new license: Within 24 hours after registering, you will receive an e-mail containing an updated license file. Follow the instructions in the e-mail to install this license file.

  For details about the support services license, please visit http://www.intel.com/software/products/compilers/cwin/pricelist.htm.

3. Obtain administrative (not power user) privilege that is required in order to install the Intel C++ Compiler correctly.
4. If you have installed an older package of the Intel C++ Compiler 8.x for Windows, uninstall it before installing the newer package.
5. Download the compiler package or purchase the product CD-ROM.
6. Run the downloaded executable or setup.exe from the CD-ROM and follow the setup program to finish the installation.
7. Use the Intel C++ Compiler at a command prompt or within Microsoft Visual C++ 6.0 or Visual C++ .NET.

Uninstalling or Repairing the Intel C++ Compiler

Administrative (not power user) privilege is also required in order to uninstall the Intel C++ Compiler correctly. Use Windows **`[Control Panel->add/remove programs]`** to uninstall the Intel C++ Compiler.

If you have installed the Intel License Manager for FLEXlm, uninstall it separately.

Using the Intel® C++ Compiler

A valid FLEXlm license must be installed before going forward.

The Intel C++ Compiler can be used from either command line or within the Visual C++ IDE. It is integrated into the Visual C++ 6.0 IDE through the "Compiler Selection Tool" and the Visual C++ .NET IDE through the Visual Studio* Integration Program (VSIP).

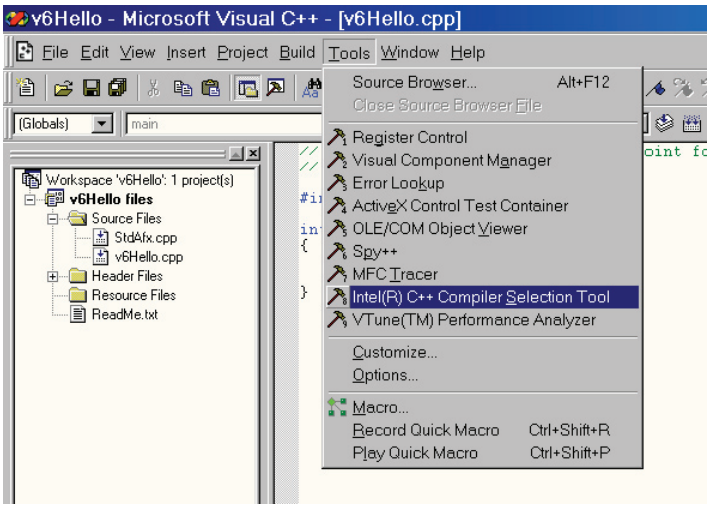
Invoking the Intel C++ Compiler from a command window:

1. Open the command window from **`Start > All programs > Intel® Software Development Tools > Intel® C++ Compiler 8.1 > Build Environment for IA-32 applications.`**
2. Invoke **icl** from the command window:

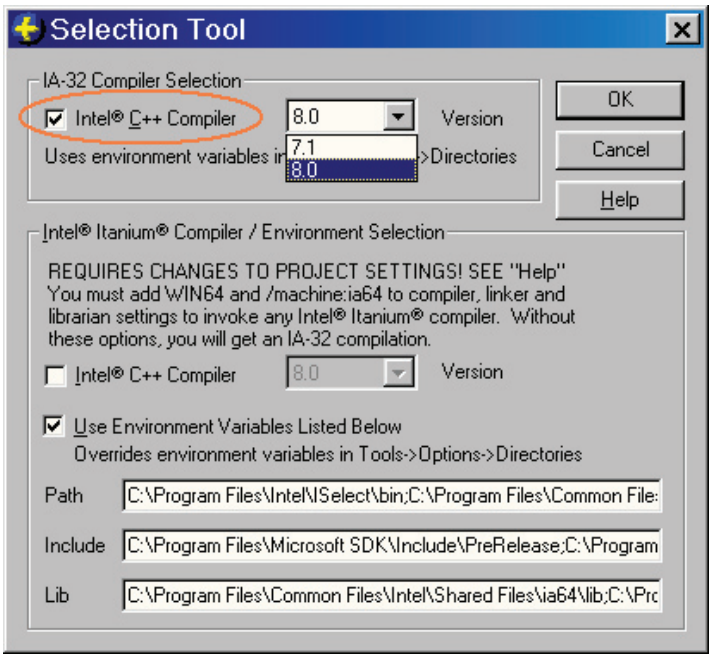
```
>> icl hello.cpp
```

Invoking the Intel C++ Compiler within Visual C++* 6.0 IDE:

1. Create a new project or open an existing project.
2. From the Tools menu, open the "**Intel® C++ Compiler Selection Tool**" dialog box menu.



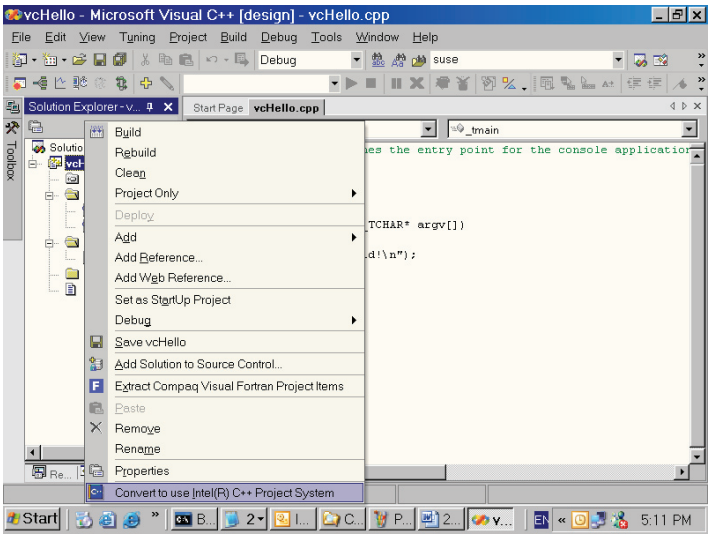
3. Check the box "**Intel® C++ Compiler**".



4. In the project settings, add any extra options specific to Intel C++ Compiler.
5. Build your application, and you should see that it uses the Intel C++ Compiler.

Invoking the Intel C++ Compiler within Visual C++ .NET IDE:

1. Create a new project or open an existing project.
2. Convert the project to use Intel C++ Project System using the pop-up menu by right clicking the project or solution:



3. In the project property dialog box, some Intel C++ Compiler-specific options have been integrated. You can also add more specific Intel C++ Compiler options here.
4. Build your application, and it should use the Intel C++ Compiler.

More details on using the Intel C++ Compiler for Windows are discussed below.

Building "Hello World" with the Intel C++ Compiler

Building "Hello World" in command line for IA-32 processors or Itanium processors

The following describes the steps to building the classic "**Hello World**" program.

1. Create a simple "Hello World" C++ program in a text editor "**hello.cpp**":

```
#include <iostream>
using namespace std;
int main()
{
cout << "Hello World!\n" << endl;
return 0;
}
```

2. Set up a command window with proper environment settings:
   - To build the program to run on an IA-32 system:
     a. Open a command window from
     [Start->Programs->Intel® Software Development Tools->Intel® C++ Compiler 8.1->Build Environment for 32-bit applications]
     b. Or open a regular command window, run the command below to set the environment settings:

```
>> <install-dir>\compiler80\ia32\bin
iclvars.bat
```

Or run the following command if the Intel C++ Compiler is installed to the default directory.

```
>> "c:\program files\intel\cpp\
compiler80\ia32\bin\iclvars.bat
```

- To build the program to run on an Itanium-based system from an IA-32 system:

  a. Open a command window from
  [Start >Programs->Intel® Software Development Tools->Intel® C++ Compiler 8.1->Build Environment for Itanium® applications]

  b. Or open a regular command window, run the command below to set the environment settings:

  ```
  >> <install-dir>\compiler80\Itanium\
  bin\iclvars.bat
  ```

  Or run the following command if the Intel C++Compiler is installed to the default directory.

  ```
  >> "c:\program files\intel\cpp\
  compiler80\Itanium\bin\iclvars.bat
  ```

3. Compile hello.cpp from the command window opened from above step:

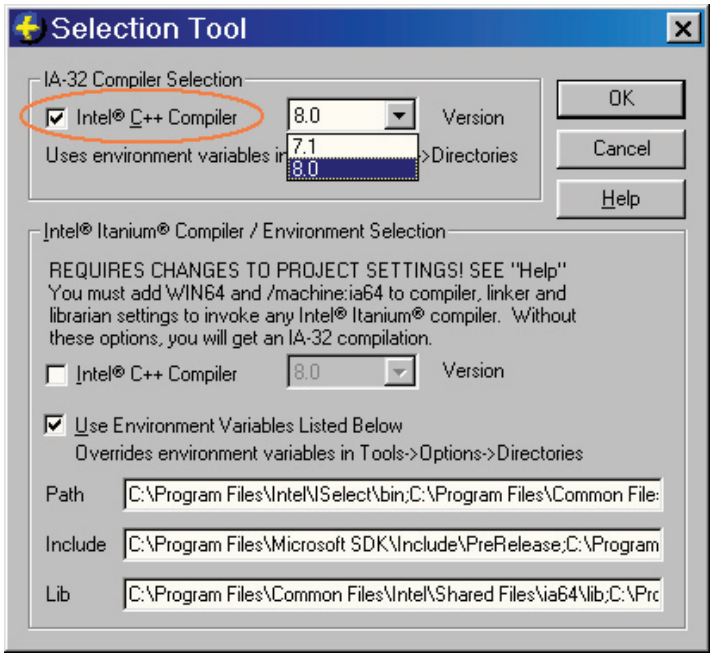
   ```
   >> icl hello.cpp
   ```

4. Run the executable on an appropriate platform, an IA-32 system or an Itanium-based system.

   ```
   >> hello.exe
   Hello World!
   >>
   ```

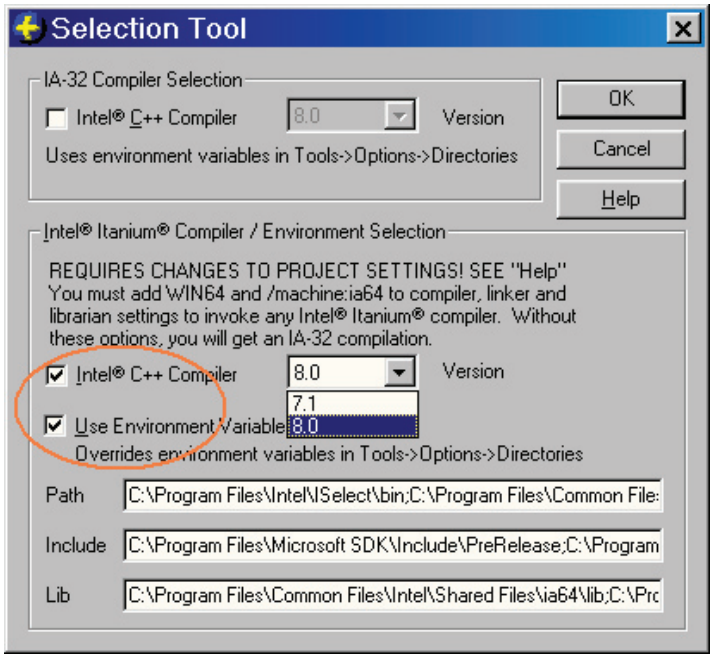
Building "Hello World" for IA-32 systems within Microsoft Visual C++* 6.0 IDE

1. Open Microsoft Visual C++ 6.0.
2. Create a **Win32 Console Project** named "**hello**"; select "**Hello World**" application when creating the project.
3. From **Tools > Intel® C++ Compiler Selection Tool**, open the "**Selection Tool**" dialog box.
4. Inside the group "**IA-32 Compiler Selection**", check "**Intel C++ Compiler**" and then click "**OK**".
5. Build the project. You'll notice the Intel C++ Compiler "**icl**" is used in the output window.
6. Run the executable to test.

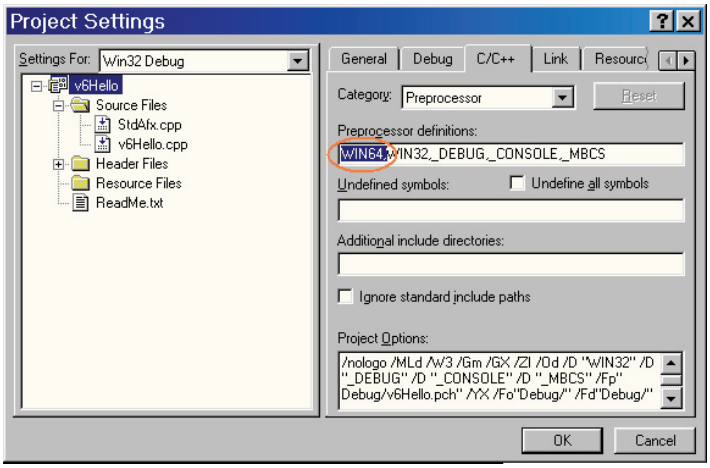


Building "Hello World" for Itanium-based systems within Microsoft Visual C++ 6.0 IDE

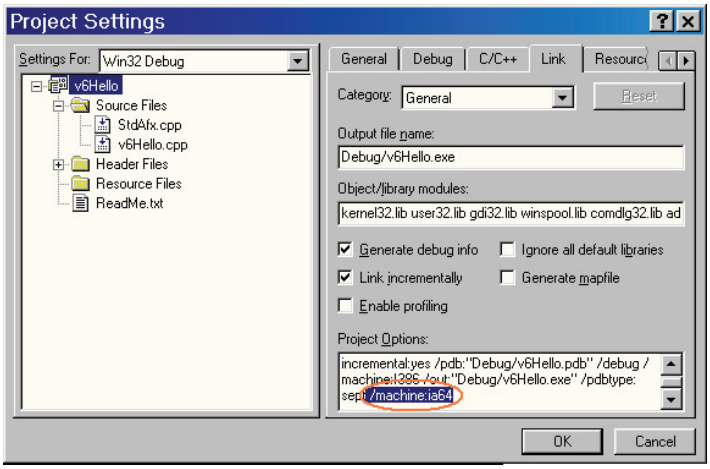
1. Follow steps 1, 2, and 3 above.
2. Inside the group "**Intel Itanium Compiler/ Environment Selection**", check "**Intel C++ Compiler**" and "**Use Environment Variables Listed Below**" and then click "**OK**".



3. Open the "**project settings**" dialog box. Click on the "**C/C++**" tab, and in the "**Preprocessor definitions**" input box, add "**WIN64**".



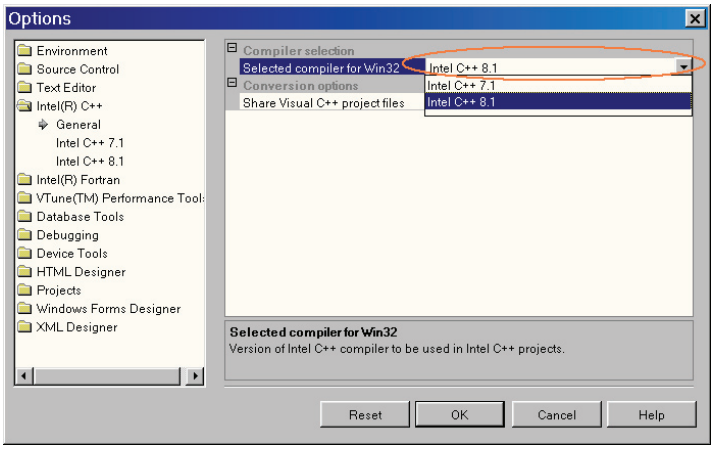
4. Click on the "**Link**" tab, and to the "**Project Options**" input box, add "**/machine:ia64**".
5. Click "**OK**".



6. Build the project. You will see that "**Intel® C++ Compiler for Itanium®-based applications**" is displayed in the output window.
7. Run the executable on an Itanium-based system to test.

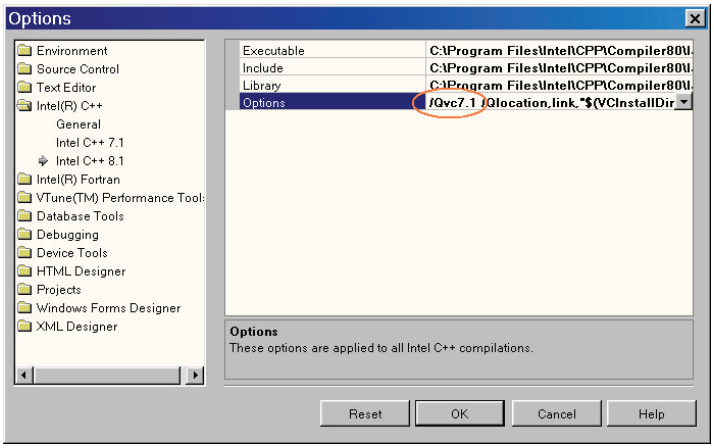
Building "Hello World" for IA-32 systems with the Intel C++ Compiler within Microsoft Visual C++ .NET 2002 or 2003 IDE

1. Create a **C++ Win32** project called "**hellow**". On the "**Application Settings**" tab, select "**Console application**" and then click the "**Finish**" button.
2. Open "**hellow.cpp**" and add the following:
   a. To the top add:
   ```
   #include <iostream>
   using namespace std;
   ```
   b. To "**main()**"add:
   ```
   cout << "Hello World!" <<
   endl;
   ```
3. Open **`Tools > Options > Intel® C++.`** In the left pane, click "**General**".
   a. In the right pane, select a version of Intel C++ Compiler you'd like to use.
   b. Then click on "**Intel C++ 7.1" or "Intel C++ 8.1**" to set the directories and default options.
   c. Then click "**OK**".



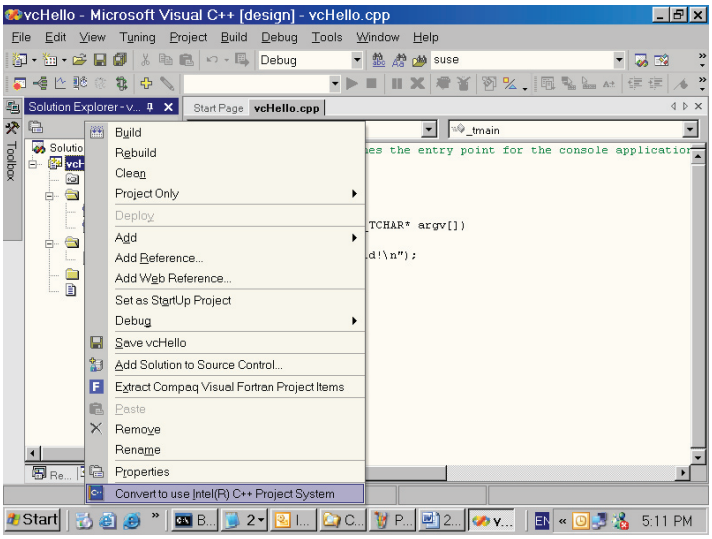
Notes:

- When using Visual C++ .NET 2003, make sure "**/Qvc7.1**" is presented in the "**Options**" edit box.
- When using Visual C++ .NET 2002, make sure "**/Qvc7**" is presented in the "**Options**" edit box.

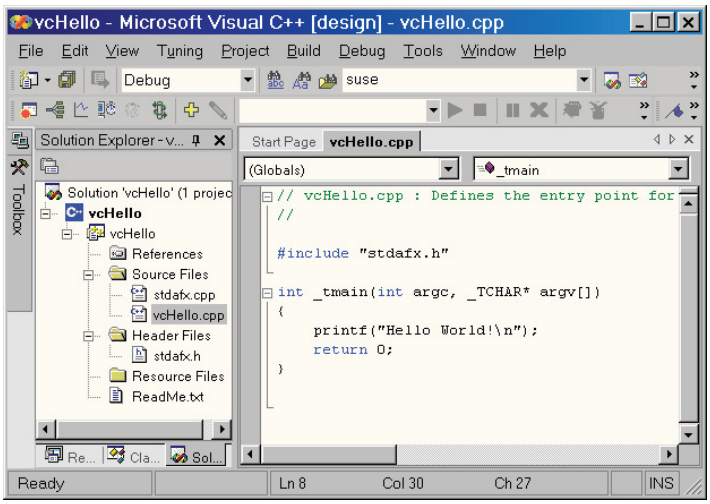


4. Convert the project to use the Intel C++ Compiler: right click on the project name "**hellow**", then do one of the following:

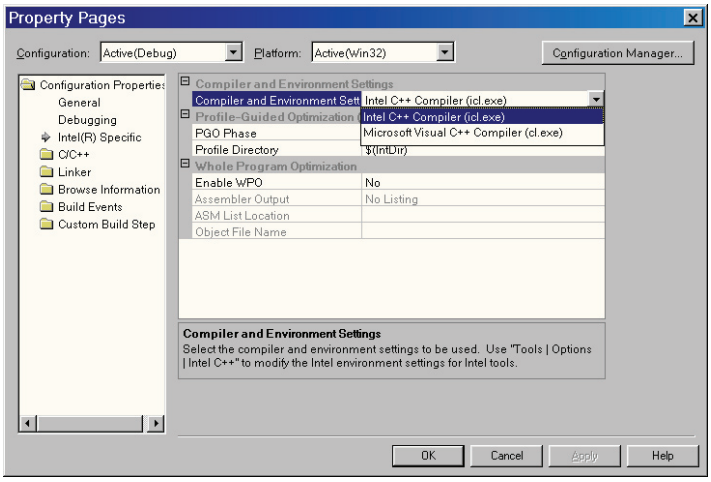
- From the pop-up menu select "**Convert to use Intel® C++ Project System**" to create a hellow.icproj file.
- Or use "**icProjConvert80**" to convert. See "The IcProjConvert80 Utility" section on how to use "**icProjConvert80**".



After converting, a new layer has been added to the "**Solution Explorer**" window. See below for detail.



5. Open "**hello Property Pages**", and click on "**Intel® Specific".**
   From the "Compiler and Environment Settings" drop-down box, select "**Intel C++ Compiler (icl)**" and click "**OK**".
6. Build the solution.
7. Run the executable to test.



Building existing Visual C++ 6.0 "Hello World" for IA-32 systems with the Intel C++ Compiler within Microsoft Visual C++ .NET 2002 or 2003 IDE

1. Open the existing "**Hello World**" project with Microsoft Visual C++ .NET and follow the prompt to convert the project into a Visual C++ .NET solution.
2. Follow steps 3, 4, 5 and 6 above to build the solution.

Building only one file in a project with the Intel C++ Compiler

Building one file with the Intel C++ Compiler within Microsoft Visual C++ 6.0

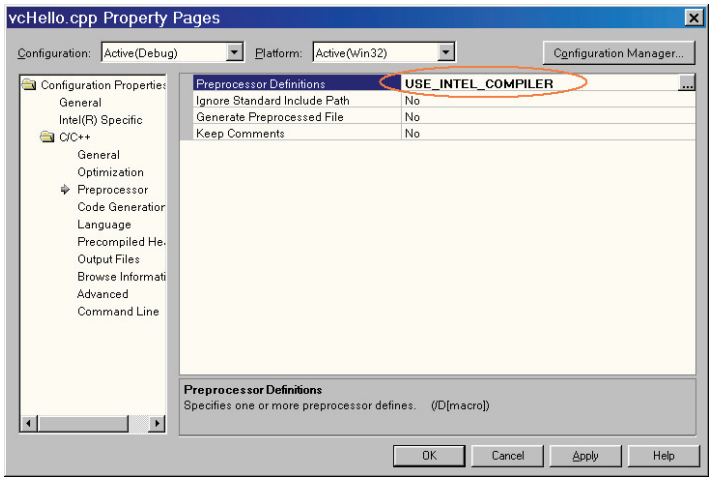
1. Open your project with Visual C++ 6.0.
2. From **Tools** menu, select **Intel® C++ Compiler Selection Tool**; The "Selection Tool" dialog box should pop up.
   - In the dialog, uncheck the boxes beside "**Intel C++ Compiler**".
3. Right click on the file you want to build with the Intel C++ Compiler, and select "**settings**" in Visual C++ 6.0.
   - In "**preprocessor definitions**", add "**_USE_INTEL_COMPILER**" and then click "**OK**".
4. Build the project. You'll notice the Intel C++ Compiler is used for the file in the output window.
5. Run the executable on an appropriate system to test.

Building one file with the Intel C++ Compiler within Microsoft Visual C++ .NET 2002 or 2003 IDE

1. Open your project with Visual C++ .NET 2002 or 2003 and open the project's property dialog box.
2. In the Solution Explorer, right-click on the project, and from the pop-up menu, select "**Convert to use Intel® C++ Project System**".

This will add another layer to the project in the Solution Explorer.

3. Open the project's **property** page:
   - In the left pane, click on "**Intel® Specific**".
   - From the drop-down list in the right pane, select "Microsoft Visual C++ Compiler **(cl.exe)**".
   - Click "**OK**".
4. Choose one of the following **two ways** to set up the system so the Intel C++ Compiler will be used for the source file:
   - Open the property page of the file that you want to compile with Intel C++ Compiler; in the left pane click on "**Intel® Specific**"; from the drop-down list in the right pane select "**Intel C++ Compiler (icl.exe)**"; click "**OK**".
   - Or open the property page of the file that you want to compile with Intel C++ Compiler; in the left pane click on "**C/C++**" and select "**Preprocessor**"; to "**Preprocessor Definitions**" add "**_USE_INTEL_COMPILER**"; click "**OK**".



5. Build the project. You'll notice the Intel C++ Compiler is used for the above file.
6. Run the executable.

Utilities in Intel C++ Compiler

The Code-Coverage Utility - codecov

This utility induces the Intel compilers' profile-guided optimization technology to present developers with a complete picture of the coverage of their application code on a particular work load. For more detailed information, please visit http://www.intel.com/software/products/compilers/techtopics/pgt.htm.

The major features include basic-block coverage, function coverage, dynamic counters, differential coverage, coverage analysis of a subset of application modules, and more. The utility also shows which blocks of code or functions are covered or not by the work loads.

The basic steps of using this utility

1. Obtain the application source code.
2. Compile with "**-Qprof_genx**" to generate a **.spi** file (the default name is "**pgopti.spi**").
3. Run the application with a good work load to create a **.dyn** file.
4. Generate a **.dpi** file in one of the following ways:
   - Recompile with "**-Qprof_use**" to generate a **.dpi** file by merging all the **.dyn** files.
   - Or use the "**profmerge**" utility, running it from the same directory as the **.dyn** file

     ```
     >> profmerge –prof_dpi test.dpi
     ```
5. Run codecov:

   ```
   >> codecov –prj myproj –spi pgopti.spi –dpi
   –pgopti.dpi
   ```
6. Check the result with any Web browser.

Usage: type "**codecov /?**" from command window for the latest description.

```
codecov [-spi spi_file] [-dpi dpi_file] [-prj proj_
name] [-counts] [-nopartial]
[-comp comp_name] [-ref ref_dpi_file] [-demang]
[-mname name] [-maddr addr]
[-bcolor name] [-fcolor name] [-pcolor name]
[-ccolor name] [-ucolor name]
```

Where:

Option	Description
-spi	Sets the filename to use for static profile information. Default is **pgopti.spi**
-dpi	Sets the filename to use for dynamic profile information. Default is **pgopti.dpi**
-prj	Sets the project name. There is no default project name.
-counts	Generates dynamic execution counts. Execution counts are not generated by default.
-nopartial	Treats partially-covered code the same as fully-covered code. By default, they are not treated the same.
-comp	Sets the filename that contains the list of files of interest. There is no default component file name.
-ref	Finds the differential coverage with respect to **ref_dpi_file**. Differential coverage is not computed by default.
-demang	Demangles both argument types of C++ functions and their names. C++ function arguments are not demangled by default on Windows.
-mname	Sets the name of the Web-page owner. For two-word names (e.g., **Your Name**), enter **Your_ Name**. Default Web-page owner name is **Nobody**.
-maddr	Sets the e-mail address of the Web-page owner. Default e-mail address of the Web-page owner is **Nobody**.
-bcolor	Sets the html color name or code of the uncovered blocks. By default, uncovered blocks are colored yellow (#ffff99).
-fcolor	Sets the html color name or code of the uncovered functions. By default, uncovered functions are colored hot pink (#ffcccc).
-pcolor	Sets the html color name or code of the partially-covered code. By default, partially-covered code is colored light brown (#fafad2).
-ccolor	Sets the html color name or code of the covered code. By default, covered code is colored white (#ffffff).
-ucolor	Sets the html color name or code of the unknown code.

Examples:

1. Basic usage:

```
>> codecov -prj myproj -spi pgopti.spi -dpi
-pgopti.dpi
```

2. To find out the differential coverage with two runs:

```
>> codecov -prj myproj -dpi customer.dpi -ref
apptest.dpi
```

3. To get code coverage for a subset of application modules, use the "**-comp**" option:

```
>> codecov -prj myproj -spi pgopti.spi -dpi
-pgopti.dpi -comp comp_filename
```

The Test-Prioritization Utility – tselect

This utility also uses the Intel compilers' profile-guided optimization technology to select and prioritize an application's tests based on prior execution profiles of the application. For more detailed information, visit http://www.intel.com/software/products/compilers/techtopics/pgt.htm.

The basic steps of using this utility

1. Obtain the application source code.
2. Compile with "**-Qprof_genx**" to generate a **.spi** file (the default name is "**pgopti.spi**").
3. Run the application with a good work load to create a **.dyn** file.
4. Generate a **.dpi** file in one of the following ways:
   - Recompile with "**-Qprof_use**" to generate the **.dpi** file by merging all the **.dyn** files.
   - Or use the "**profmerge**" utility, running it from the same directory as the **.dyn** file

     ```
     >> profmerge -prof_dpi test.dpi
     ```
5. Repeat steps 3 and 4 for all your test suites.
6. Create a text file "**dpilistfile.txt**" that contains all the **.dpi** files generated above; each line lists only one **.dpi** file.
7. Run **tselect**:

   ```
   >> tselect -dpi_list dpilistfile.txt -spi
   pgopti.spi
   ```
8. Check the result with any Web browser.

Usage:

```
tselect -dpi_list <DPI_list file>
[-spi <SPI file>] [-o <report file>]
[-comp <component file>] [-cutoff <value>]
[-mintime] [-nototal] [-verbose]
```

Where:

Option	Description
-dpi_list <DPI_list file>	Each line of **<DPI_list file>** should include the name of one DPI file.
-spi <SPI file>	**<SPI file>** is the path to the main SPI file.
-o <report file>	**<report file>** is the path to the generated report file.
-comp <component file>	**<component file>** is the path to the component file. Each line of **<component file>** should include one module or dir name, for example: **/dev/src, c:\dev\src, src.c, src.h** are all valid names.
-cutoff <value>	Terminate when we reach **<value>%** of precomputed total coverage. **<value>** must be greater than 0.0, e.g., **99.00**.
-mintime	Minimize testing execution time; execution time of each test must be provided on the same line of **DPI_list** file after the test name in **dd:hh:mm:ss** format.
-nototal	Do not precompute the total coverage.
-verbose	Enable more logging information about the program progress.

The icProjConvert80 Utility

This utility is used to convert a solution or project(s) in a solution from the Visual C++ .NET project system to the Intel C++ Compiler project system, or vice versa.

Usage:

```
ICProjConvert80 [sln_file] [prj_files] </VC or /IC>
[/s] [/sharedvcproj] [/nologo]
```

Where:

Option	Description
sln_file	A path to a solution file, which should be converted to the specified project system. This is the **.sln** file name.
prj_files	A space separated list of project names, which should be converted to the specified project system. This is not the **.vcproj** file name, it's the project name in the Solution Explorer.
/VC	Convert to the Visual C++ .NET project system.
/IC	Convert to the Intel C++ project system.
/s	Silent mode – all information messages (except errors) are hidden.
/nologo	Suppress the startup banner.
-nototal	Do not precompute the total coverage.
/sharedvcproj	Create shared **vcproj**.
/? or /h	Show help.

Examples:

1. Convert to use the Intel C++ Compiler project system.
   >> icProjConvert80 hellow.sln /IC
   Convert all projects in a solution.
   >> icProjConvert80 hellow /IC
   Convert the project "hellow" only.
2. Convert back to use the Visual C++ .NET project system.
   >> icProjConvert80 hellow.sln /VC
   Convert all projects in a solution.
   >> icProjConvert80 hellow.sln test /VC
   Convert the project "test" in a solution.

Using the "Share Visual C++ project files" feature

The feature "**Share Visual C++ project files**" means to share the "**.vcproj**" files among developers or among different applications.

There are two situations where this feature comes in really handy:

- When working with many developers to create an application and there's only one copy of the Intel C++ Compiler, you can use this option to share the "**.vcproj**" among the developers without conflicts. Everybody who shares the Intel C++ Compiler should maintain their own "**.sln**" file because this file will be modified when converting the project(s) to use the Intel C++ project system, but the "**.vcproj**" will not be modified. So the "**.vcproj**" files can be shared.
- Whenever you develop a library for multiple applications or groups, and only one application or one group uses the Intel C++ Compiler, you should use this feature in this case too. The same rule about the "**.sln**" applies here.

Note:
When you convert your project or solution to use the Intel C++ project system, a new file "**.icproj**" will be created for each "**.vcproj**" file. This "**.icproj**" file contains all the options that are specific to the Intel C++ Compiler like "**-Qipo**", "**-QxN**" or "**-Qprof_gen**".

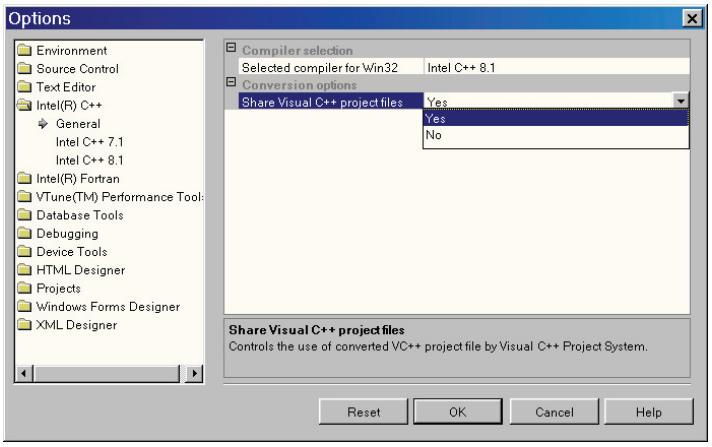
There are two methods to achieve the goal above:

- Converting your project from command line:

```
>> ICProjConvert8x.exe myproj -ic -sharevcproj
```

- Converting your project within the Visual C++ .NET IDE:

Before converting your project from the pop-up menu, open the Options dialog and select the "**Intel C++**" page and click on "**General**". Make sure that the "Share Visual C++ project files" is set to "Yes". In Version 8.1 of the Intel C++ Compiler, this field is set to "**Yes**" as the default.



Utility to enable a power user other than the administrative user to use the Intel selection tool within Visual C++ 6.0

This utility is located at **Start > Programs > Intel® Software Development Tools > Intel® C++ Compiler 8.1 > Update User's Registry**. For a power user to use the Intel selection tool within Visual C++ 6.0 IDE, run Visual C++ 6.0 once, then this utility.

Compatibility with Visual C++* .NET

The Intel C++ Compiler 8.1 supports limited features of Visual C++ .NET, and does not support the attribute feature and managed code.

For more detailed information, please refer to "Intel C++ Compiler 8.1 and Microsoft Visual C++ .NET Compatibility" at http://www.intel.com/software/products/compilers/cwin/.

For more information on using Intel C++ Compiler 8.1 within Visual C++ .NET, please see "Using the Intel® C++ Compiler with Microsoft Visual C++ .NET" section in the *Intel C++ Compiler User's Guide* at **<install-dir>\compiler80\docs\ccug.chm.**

Getting Started with Intel® Compiler Optimizations

The Intel C++ Compiler enables programmers to take full advantage of the advanced performance enhancement features of Intel's latest IA-32 and Itanium processors, as well as advanced optimizations. These include support for Streaming SIMD Extensions 3, profile-guided optimization, interprocedural optimization, vectorization and processor dispatch.

The optimizations are intended for use in product-release builds of applications, not necessarily for earlier phases of application development cycles. In general, increasing the degree of optimization done by the compiler leads to an increase in compile-time and reduced debugging capability. This section describes an optimization methodology with the Intel C++ Compiler.

During the application development, the "**-Zi -Od**" switches are recommended to allow fast compile times and full debugging with no optimization. To start to optimize, the default optimization "**-O2**" is recommended. The "**-O3**" option enables advanced optimizations. The "**-QaxN**", "**-QaxP**", "**-QaxB**", "**-QaxW**", "**-QaxK**", "**-QaxM**" and "**-Qaxi**" switches are used for generating specialized code for specific Intel® processors as well as generic IA-32 code. **Interprocedural optimization** allows the compiler to optimize across different compilation units and can provide large performance improvements. **Profile guided optimization** uses information obtained by running an instrumented executable that allows the compiler to rebuild the application knowing where the majority of the computations are. Of course, not all optimizations are beneficial for all applications. For additional details on optimizing, the paper, "Optimizing Applications with the Intel C++ and Fortran Compilers" is available at http://www.intel.com/software/products/compilers/cwin/. For complete information on the individual optimizations, please refer to the *Intel C++ Compiler's Compiler User's Guide* at **`<install-dir>\compiler80\docs\ccug.chm`**.

Remember to always measure the performance of your application after each optimization to verify the benefits. The VTune™ Performance Analyzer can be a great help for measuring the performance benefits of each, as well as giving advice on further tuning opportunities. Additional information is available at http://www.intel.com/software/products/vtune/.

The Intel® C++ Compiler, Extended Memory 64 Technology Edition for Windows*

The Intel® C++ Compiler, Extended Memory 64 Technology (EM64T) Edition is designed to create optimized applications for Intel® EM64T systems. Refer to its release notes for the system requirements, installation notes and the latest updates.

The Intel C++ Compiler EM64T Edition is an IA-32 application and it can be installed on any IA-32-based system or Intel EM64T-based system running Windows. Integration into the Microsoft Visual C++ .NET development environment is not provided in this release. Use the command-line environment to build applications for Intel EM64T-based systems.

The Intel Debugger for Intel EM64T systems is part of this compiler package, but only command-line interface is supported in this release. It only supports native debugging functionality; remote-debugging is not supported in this release.

Additional Information

Intel® Software College provides a one-stop shop at Intel for training developers on leading edge software development technologies. Training consists of online and instructor-led courses covering all Intel architectures, platforms and technologies. You can find more information about it at http://www.intel.com/software/college/.

Intel® Premier Support Web site. Your feedback is very important to us. To receive technical support and product updates for the tools provided in this product you need to register at http://www.intel.com/software/products/registrationcenter/. To submit an issue or feature request, please go to Intel Premier Support at https://premier.intel.com/ after registering.

Compiler support information: Top technical issues and product errata are available at http://support.intel.com/support/performancetools/c/windows.

Product release notes: Located at **`<install-dir>\compiler80\docs\Crelnotes.htm`**

Compiler User's Guide: Located at **`<install-dir>\compiler80\docs\ccug.chm`**

Bug fix information: For each package of the Intel compiler, there is a text file posted at https://premier.intel.com/. The file name is something like **<package_id>_README.txt**. It contains the bug fix information for the package.

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

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

304278-001