IBM XL C/C++ for Linux, V13.1

Getting Started with XL C/C++

Version 13.1

SC27-4247-00

IBM XL C/C++ for Linux, V13.1

Getting Started with XL C/C++

Version 13.1

SC27-4247-00

Note

Before using this information and the product it supports, read the information in "Notices" on page 59.

First edition

This edition applies to IBM XL C/C++ for Linux, V13.1 (Program 5765-J08; 5725-C73) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

© Copyright IBM Corporation 1996, 2014.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this document v
Conventions v
Related information ix
IBM XL C/C++ information ix
Standards and specifications x
Other IBM information. xi
Other information xi
Technical support xi
How to send your comments xi

Chapter 1. Introducing XL C/C++ 1
Commonality with other IBM compilers 1
Operating system support 1
A highly configurable compiler 1
Language standard compliance 3
Compatibility with GNU 3
Source-code migration and conformance checking 4
Libraries 4
Tools, utilities, and commands 5
Program optimization 7
64-bit object capability 7
Shared memory parallelization 8
Diagnostic listings 8
Symbolic debugger support 9

Chapter 2. What's new for IBM XL C/C++ for Linux, V13.1 11
Support for POWER8 processors 11
Advance Toolchain 7.0 support 12
C++11 features 12
C11 features 13
OpenMP 4.0 14
Built-in functions 14
Compiler options and pragma directives 18
Performance and optimization 20

Chapter 3. Migrating from earlier versions 21
Enhancements added in Version 12.1 21
C++11 features 21
C11 features 24
OpenMP 3.1 25
Performance and optimization 25
Diagnostic reports 26
Built-in functions 27
Compiler options and pragma directives . . . 27
Enhancements added in Version 11.1 30
Support for POWER7 processors 30
C++11 features 31
Performance and optimization 34
New diagnostic reports 36
Utilization tracking and reporting tool 38
New or changed compiler options and directives 38
Built-in functions 42
Enhancements added in Version 10.1 43
C++11 features 43
Other XL C/C++ language-related updates . . . 45
OpenMP 3.0 45
Performance and optimization 46
Compiler options and pragma directives . . . 47
Predefined macros 48

Chapter 4. Setting up and customizing XL C/C++. 49
Using custom compiler configuration files 49
Configuring compiler utilization tracking and reporting 49

Chapter 5. Developing applications with XL C/C++ 51
The compiler phases 51
Editing C/C++ source files 51
Compiling with XL C/C++ 52
Invoking the compiler 52
Compiling parallelized XL C/C++ applications 52
Specifying compiler options 53
XL C/C++ input and output files 54
Linking your compiled applications with XL C/C++ 54
Dynamic and static linking 55
Running your compiled application 55
XL C/C++ compiler diagnostic aids 56
Debugging compiled applications 56
Determining which level of XL C/C++ is being used 57

Notices 59
Trademarks and service marks 61

Index 63

© Copyright IBM Corp. 1996, 2014

About this document

This document contains overview and basic usage information for the IBM® XL C/C++ for Linux, V13.1 compiler.

Who should read this document

This document is intended for C and C++ developers who are looking for introductory overview and usage information for XL C/C++. It assumes that you have some familiarity with command-line compilers, a basic knowledge of the C and C++ programming languages, and basic knowledge of operating system commands. Programmers new to XL C/C++ can use this document to find information on the capabilities and features unique to XL C/C++.

How to use this document

Unless indicated otherwise, all of the text in this reference pertains to both C and C++ languages. Where there are differences between languages, these are indicated through qualifying text and icons, as described in "Conventions."

Throughout this document, the **xlc** and **xlc++** compiler invocations are used to describe the actions of the compiler. You can, however, substitute other forms of the compiler invocation command if your particular environment requires it, and compiler option usage will remain the same unless otherwise specified.

While this document covers information on configuring the compiler environment, and compiling and linking C or C++ applications using the XL C/C++ compiler, it does not include the following topics:

- Compiler installation: see the *XL C/C++ Installation Guide* for information on installing XL C/C++.
- Compiler options: see the *XL C/C++ Compiler Reference* for detailed information on the syntax and usage of compiler options.
- The C or C++ programming languages: see the *XL C/C++ Language Reference* for information on the syntax, semantics, and IBM implementation of the C or C++ programming languages.
- Programming topics: see the *XL C/C++ Optimization and Programming Guide* for detailed information on developing applications with XL C/C++, with a focus on program portability and optimization.

Conventions

Typographical conventions

© Copyright IBM Corp. 1996, 2014

The following table shows the typographical conventions used in the IBM XL C/C++ for Linux, V13.1 information.

Table 1. Typographical conventions

Typeface	Indicates	Example
bold	Lowercase commands, executable names, compiler options, and directives.	The compiler provides basic invocation commands, **xlc** and **xlC** (**xlc++**), along with several other compiler invocation commands to support various C/C++ language levels and compilation environments.
italics	Parameters or variables whose actual names or values are to be supplied by the user. Italics are also used to introduce new terms.	Make sure that you update the *size* parameter if you return more than the *size* requested.
underlining	The default setting of a parameter of a compiler option or directive.	nomaf | maf
`monospace`	Programming keywords and library functions, compiler builtins, examples of program code, command strings, or user-defined names.	To compile and optimize myprogram.c, enter: `xlc myprogram.c -O3`.

Qualifying elements (icons)

Most features described in this information apply to both C and C++ languages. In descriptions of language elements where a feature is exclusive to one language, or where functionality differs between languages, this information uses icons to delineate segments of text as follows:

Table 2. Qualifying elements

Qualifier/Icon	Meaning
C only, or C only begins C C C only ends	The text describes a feature that is supported in the C language only; or describes behavior that is specific to the C language.
C++ only, or C++ only begins C++ C++ C++ only ends	The text describes a feature that is supported in the C++ language only; or describes behavior that is specific to the C++ language.
IBM extension, or IBM extension begins IBM IBM IBM extension ends	The text describes a feature that is an IBM extension to the standard language specifications.

Table 2. Qualifying elements (continued)

Qualifier/Icon	Meaning
C11, or C11 begins ▶ C11 C11 ◀ C11 ends	The text describes a feature that is introduced into standard C as part of C11.
C++11, or C++11 begins ▶ C++11 C++11 ◀ C++11 ends	The text describes a feature that is introduced into standard C++ as part of C++11.

Syntax diagrams

Throughout this information, diagrams illustrate XL C/C++ syntax. This section will help you to interpret and use those diagrams.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►── symbol indicates the beginning of a command, directive, or statement.

  The ──► symbol indicates that the command, directive, or statement syntax is continued on the next line.

  The ►── symbol indicates that a command, directive, or statement is continued from the previous line.

  The ──►◄ symbol indicates the end of a command, directive, or statement.

  Fragments, which are diagrams of syntactical units other than complete commands, directives, or statements, start with the |── symbol and end with the ──| symbol.
- Required items are shown on the horizontal line (the main path):

```
►►──keyword──required_argument──────────────────────────────────────────►◄
```

- Optional items are shown below the main path:

```
►►──keyword──┬──────────────────┬──────────────────────────────────────►◄
             └─optional_argument─┘
```

- If you can choose from two or more items, they are shown vertically, in a stack.

  If you *must* choose one of the items, one item of the stack is shown on the main path.

```
►►──keyword──┬─required_argument1─┬──────────────────────────────────►◄
             └─required_argument2─┘
```

  If choosing one of the items is optional, the entire stack is shown below the main path.

```
►►──keyword──┬──────────────────┬──────────────────────────────────────►◄
             ├─optional_argument1─┤
             └─optional_argument2─┘
```

- An arrow returning to the left above the main line (a repeat arrow) indicates that you can make more than one choice from the stacked items or repeat an item. The separator character, if it is other than a blank, is also indicated:

```
             ┌─,──────────────────┐
             ▼                    │
►►──keyword──┴─repeatable_argument─┴──────────────────────────────────────►◄
```

- The item that is the default is shown above the main path.

```
             ┌─default_argument───┐
►►──keyword──┴─alternate_argument─┴──────────────────────────────────────►◄
```

- Keywords are shown in nonitalic letters and should be entered exactly as shown.
- Variables are shown in italicized lowercase letters. They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Sample syntax diagram

The following syntax diagram example shows the syntax for the **#pragma comment** directive.

```
  (1)    (2)        (3)        (4)    (5)                                    (9)  (10)
►►──────#──────pragma──────comment──────(──┬─compiler──────────────────────────┬──)──────────────►◄
                                           ├─date────────────────────────────┤
                                           ├─timestamp───────────────────────┤
                                           │          (6)                    │
                                           │┌─copyright─┐┌──────────────────┐│
                                           │└─user──────┘│ (7)          (8) │
                                           └─────────────└─,──────"─token_sequence─"──┘
```

Notes:

1 This is the start of the syntax diagram.

2 The symbol # must appear first.

3 The keyword `pragma` must appear following the # symbol.

4 The name of the pragma `comment` must appear following the keyword `pragma`.

5 An opening parenthesis must be present.

6 The comment type must be entered only as one of the types indicated: `compiler`, `date`, `timestamp`, `copyright`, or `user`.

7 A comma must appear between the comment type `copyright` or `user`, and an optional character string.

8 A character string must follow the comma. The character string must be enclosed in double quotation marks.

9 A closing parenthesis is required.

10 This is the end of the syntax diagram.

The following examples of the **#pragma comment** directive are syntactically correct according to the diagram shown above:

```
#pragma comment(date)
#pragma comment(user)
#pragma comment(copyright,"This text will appear in the module")
```

Examples in this information

The examples in this information, except where otherwise noted, are coded in a simple style that does not try to conserve storage, check for errors, achieve fast performance, or demonstrate all possible methods to achieve a specific result.

The examples for installation information are labelled as either *Example* or *Basic example*. *Basic examples* are intended to document a procedure as it would be performed during a basic, or default, installation; these need little or no modification.

Related information

The following sections provide related information for XL C/C++:

IBM XL C/C++ information

XL C/C++ provides product information in the following formats:

- README files

  README files contain late-breaking information, including changes and corrections to the product information. README files are located by default in the XL C/C++ directory and in the root directory of the installation CD.
- Installable man pages

  Man pages are provided for the compiler invocations and all command-line utilities provided with the product. Instructions for installing and accessing the man pages are provided in the *IBM XL C/C++ for Linux, V13.1 Installation Guide*.
- Information center

  The fully searchable HTML-based documentation is viewable on the web at http://www.ibm.com/support/knowledgecenter/SSXVZZ_13.1.0/com.ibm.compilers.linux.doc/welcome.html.
- PDF documents

  PDF documents are located by default in the /opt/ibm/xlC/13.1.0/doc/*LANG*/pdf/ directory, where *LANG* is one of en_US, zh_CN, or ja_JP. The PDF files are also available on the web at http://www.ibm.com/support/docview.wss?uid=swg27036675.

  The following files comprise the full set of XL C/C++ product information:

Table 3. XL C/C++ PDF files

Document title	PDF file name	Description
IBM XL C/C++ for Linux, V13.1 Installation Guide, SC27-4248-00	install.pdf	Contains information for installing XL C/C++ and configuring your environment for basic compilation and program execution.
Getting Started with IBM XL C/C++ for Linux, V13.1, SC27-4247-00	getstart.pdf	Contains an introduction to the XL C/C++ product, with information on setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.

Table 3. XL C/C++ PDF files (continued)

Document title	PDF file name	Description
IBM XL C/C++ for Linux, V13.1 Compiler Reference, SC27-4249-00	compiler.pdf	Contains information about the various compiler options, pragmas, macros, environment variables, and built-in functions, including those used for parallel processing.
IBM XL C/C++ for Linux, V13.1 Language Reference, SC27-4250-00	langref.pdf	Contains information about the C and C++ programming languages, as supported by IBM, including language extensions for portability and conformance to nonproprietary standards.
IBM XL C/C++ for Linux, V13.1 Optimization and Programming Guide, SC27-4251-00	proguide.pdf	Contains information on advanced programming topics, such as application porting, interlanguage calls with Fortran code, library development, application optimization and parallelization, and the XL C/C++ high-performance libraries.

To read a PDF file, use Adobe Reader. If you do not have Adobe Reader, you can download it (subject to license terms) from the Adobe website at http://www.adobe.com.

More information related to XL C/C++ including IBM Redbooks® publications, white papers, tutorials, documentation errata, and other articles, is available on the web at:

http://www.ibm.com/support/docview.wss?uid=swg27036675

Note: Documentation errata is reflected only in the English version of the information center.

For more information about boosting performance, productivity, and portability, see the C/C++ café at http://www.ibm.com/software/rational/cafe/community/ccpp.

Standards and specifications

XL C/C++ is designed to support the following standards and specifications. You can refer to these standards for precise definitions of some of the features found in this information.

- *Information Technology - Programming languages - C, ISO/IEC 9899:1990*, also known as *C89*.
- *Information Technology - Programming languages - C, ISO/IEC 9899:1999*, also known as *C99*.
- *Information Technology - Programming languages - C, ISO/IEC 9899:2011*, also known as *C11*. (Partial support)
- *Information Technology - Programming languages - C++, ISO/IEC 14882:1998*, also known as *C++98*.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2003*, also known as *Standard C++*.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2011*, also known as *C++11*. (Partial support)
- *Information Technology - Programming languages - Extensions for the programming language C to support new character data types, ISO/IEC DTR 19769*. This draft

technical report has been accepted by the C standards committee, and is available at http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1040.pdf

- *Draft Technical Report on C++ Library Extensions, ISO/IEC DTR 19768*. This draft technical report has been submitted to the C++ standards committee, and is available at http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2005/n1836.pdf.
- *AltiVec Technology Programming Interface Manual*, Motorola Inc. This specification for vector data types, to support vector processing technology, is available at http://www.freescale.com/files/32bit/doc/ref_manual/ALTIVECPIM.pdf.
- *ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985*.
- *OpenMP Application Program Interface Version 4.0 (Partial support)*, available at http://www.openmp.org

Other IBM information

- *ESSL for AIX V5.1/ESSL for Linux on POWER V5.1 Guide and Reference* available at the Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL web page.

Other information

- *Using the GNU Compiler Collection* available at http://gcc.gnu.org/onlinedocs

Technical support

Additional technical support is available from the XL C/C++ Support page at http://www.ibm.com/support/entry/portal/overview/software/rational/xl_c~c++_for_linux. This page provides a portal with search capabilities to a large selection of Technotes and other support information.

If you cannot find what you need, you can send email to compinfo@ca.ibm.com.

For the latest information about XL C/C++, visit the product information site at http://www.ibm.com/software/products/us/en/xlcpp-linux/.

How to send your comments

Your feedback is important in helping to provide accurate and high-quality information. If you have any comments about this information or any other XL C/C++ information, send your comments by email to compinfo@ca.ibm.com.

Be sure to include the name of the manual, the part number of the manual, the version of XL C/C++, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

Chapter 1. Introducing XL C/C++

IBM XL C/C++ for Linux, V13.1 is an advanced, high-performance compiler that can be used for developing complex, computationally intensive programs, including interlanguage calls with C and Fortran programs.

This section contains information about the features of the XL C/C++ compiler at a high level. It is intended for people who are evaluating the compiler, and for new users who want to find out more about the product.

Commonality with other IBM compilers

IBM XL C/C++ for Linux, V13.1 is part of a larger family of IBM C, C++, and Fortran compilers. XL C/C++, together with XL Fortran, comprises the family of XL compilers.

These compilers are derived from a common code base that shares compiler function and optimization technologies for a variety of platforms and programming languages. Programming environments include IBM AIX®, IBM Blue Gene®/P, IBM Blue Gene®/Q, IBM i, selected Linux distributions, IBM z/OS®, and IBM z/VM®. The common code base, along with compliance with international programming language standards, helps support consistent compiler performance and ease of program portability across multiple operating systems and hardware platforms.

Operating system support

This section describes the operating systems that IBM XL C/C++ for Linux, V13.1 supports.

IBM XL C/C++ for Linux, V13.1 supports the following operating systems:

- SUSE Linux Enterprise Server 11 Service Pack 2 (SLES 11 SP2) or later
- Red Hat Enterprise Linux 6.4 (RHEL 6.4) or later
- Red Hat Enterprise Linux 7.0 (RHEL 7.0) or later

See the README file and "Before installing XL C/C++" in the *XL C/C++ Installation Guide* for a complete list of requirements.

IBM XL C/C++ for Linux, V13.1 is supported on big-endian systems. The compiler, its libraries, and its generated object programs run on POWER5, POWER5+, POWER6®, POWER7®, POWER7+™, and POWER8™ systems with the required software and disk space.

To exploit the various supported hardware configurations, the compiler provides options to tune the performance of applications according to the hardware type that runs the compiled applications.

A highly configurable compiler

You can use a variety of compiler invocation commands and options to tailor the compiler to your unique compilation requirements.

© Copyright IBM Corp. 1996, 2014

Compiler invocation commands

XL C/C++ provides several commands to invoke the compiler, for example, **xlC**, **xlc++**, and **xlc**. Each invocation command is unique in that it instructs the compiler to tailor compilation output to meet a specific language level specification. Compiler invocation commands are provided to support all standardized C/C++ language levels, and many popular language extensions as well.

The compiler also provides corresponding "**_r**" versions of most invocation commands, for example, **xlc_r** and **xlC_r**. The "**_r**" invocations instruct the compiler to link and bind object files to thread safe components and libraries, and produce thread safe object code for compiler-created data and procedures.

For more information about XL C/C++ compiler invocation commands, see "Invoking the compiler" in the *XL C/C++ Compiler Reference*.

Compiler options

You can choose from a large selection of compiler options to control compiler behavior. You can benefit from using different options for the following tasks:

- Debugging your applications
- Optimizing and tune application performance
- Selecting language levels and extensions for compatibility with nonstandard features and behaviors that are supported by other C or C++ compilers
- Performing many other common tasks that would otherwise require changing the source code

You can specify compiler options through a combination of environment variables, compiler configuration files, command line options, and compiler directive statements embedded in your program source.

For more information about XL C/C++ compiler options, see "Compiler options reference" in the *XL C/C++ Compiler Reference*.

Custom compiler configuration files

The installation process creates a default compiler configuration file containing stanzas that define compiler option default settings.

If you frequently specify compiler option settings other than the default settings of XL C/C++, you can use makefiles to define your settings. Alternatively, you can create custom configuration files to define your own frequently used option settings.

For more information about using custom compiler configuration files, see “Using custom compiler configuration files” on page 49.

Utilization tracking configuration file

The utilization and reporting tool can be used to detect whether your organization's use of the compiler exceeds your license entitlements.

The utilization tracking and reporting feature of the compiler has its own configuration file. The main compiler configuration file contains an entry that

points to this file. The different installations of the compiler product can use a single utilization tracking configuration file to centrally manage the utilization tracking and reporting feature.

For detailed information about the utilization tracking and reporting feature, see "Tracking and reporting compiler usage" in the *XL C/C++ Compiler Reference*.

Language standard compliance

IBM XL C/C++ for Linux, V13.1 supports the following C/C++ programming language specifications.

C language specifications

- Partial support for ISO/IEC 9899:2011 (referred to as C11)
- ISO/IEC 9899:1999 (referred to as C99)
- ISO/IEC 9899:1990 (referred to as C89)

C++ language specifications

- Partial support for ISO/IEC 14882:2011 (referred to as C++11)
- ISO/IEC 14882:2003 (referred to as Standard C++)
- ISO/IEC 14882:1998, the first official specification of the language (referred to as C++98)

In addition to the standard language levels, XL C/C++ supports the following language extensions:

- Partial support for OpenMP Application Program Interface V4.0
- OpenMP Application Program Interface V3.1
- Language extensions to support vector programming
- A subset of GNU C and C++ language extensions

See "Language levels and language extensions" in the *XL C/C++ Language Reference* for more information about C/C++ language specifications and extensions.

Compatibility with GNU

XL C/C++ supports a subset of the GNU compiler command options to facilitate porting applications that are developed with the **gcc** and **g++** compilers.

This support is available when the **gxlc** or **gxlc++** invocation command is used together with select GNU compiler options. Where possible, the compiler maps GNU options to their XL C/C++ compiler option counterparts before invoking the compiler.

The invocation commands use a plain text configuration file to control GNU-to-XL C/C++ option mappings and defaults. You can customize this configuration file to meet your unique compilation requirements. For more information, see "Reusing GNU C/C++ compiler options with gxlc and gxlc++" in the *XL C/C++ Compiler Reference*.

XL C/C++ uses GNU C and GNU C++ header files together with the GNU C and C++ runtime libraries to produce code that is binary-compatible with that produced by the GNU Compiler Collection (GCC). Portions of an application can be built with XL C/C++ and combined with portions built with GCC to produce an application that behaves as if it had been built solely with GCC.

To achieve binary compatibility with GCC-compiled code, a program compiled with XL C/C++ includes the same headers as those used by a GNU compiler residing on the same system. To ensure that the proper versions of headers and runtime libraries are present on the system, the prerequisite GCC compiler must be installed before installing XL C/C++.

Some additional noteworthy points about this relationship are:

- IBM built-in functions coexist with GNU C built-ins.
- Compilation of C and C++ programs uses the GNU C and GNU C++ header files.
- Compilation uses the GNU assembler for assembler input files.
- Compiled C code is linked to the GNU C runtime libraries.
- Compiled C++ code is linked to the GNU C and GNU C++ runtime libraries.
- Debugging uses the GNU debugger, **gdb**

Source-code migration and conformance checking

XL C/C++ provides compiler invocation commands that instruct the compiler to compile your application code to a specific language level.

You can also use the **-qlanglvl** compiler option to specify a language level. If the language or language extension elements in your program source do not conform to the specified language level, the compiler issues diagnostic messages.

See -qlanglvl in the *XL C/C++ Compiler Reference* for more information.

Libraries

XL C/C++ includes a runtime environment containing a number of libraries.

Mathematical Acceleration Subsystem library

The Mathematical Acceleration Subsystem (MASS) library consists of scalar and vector mathematical built-in functions tuned specifically for optimum performance on supported processor architectures. You can choose a MASS library to support high-performance computing on a broad range of processors, or you can select a library tuned to support a specific processor family.

The MASS library functions support both 32-bit and 64-bit compilation modes and offer improved performance over the default `libm` math library routines. These libraries are threadsafe, and are called automatically when you request specific levels of optimization for your application. You can also make explicit calls to MASS library functions, regardless of whether optimization options are in effect or not.

For more information, see "Using the Mathematical Acceleration Subsystem" in the *XL C/C++ Optimization and Programming Guide*.

Basic Linear Algebra Subprograms

The Basic Linear Algebra Subprograms (BLAS) set of high-performance algebraic functions are shipped in the libxlopt library. You can use these functions to:

- Compute the matrix-vector product for a general matrix or its transpose.

- Perform combined matrix multiplication and addition for general matrices or their transposes.

For more information about using the BLAS functions, see "Using the Basic Linear Algebra Subprograms" in the *XL C/C++ Optimization and Programming Guide*.

Other libraries

The following libraries are also shipped with XL C/C++:

- The SMP runtime library supports both explicit and automated parallel processing. See "SMP Runtime Library" in the *XL C/C++ Optimization and Programming Guide*.
- XL C++ Runtime Library contains support routines needed by the compiler.

Support for Boost libraries

IBM XL C/C++ for Linux, V13.1 provides partial support for the Boost V1.55.0 libraries. A patch file is available that modifies the Boost V1.55.0 libraries so that they can be built and used with XL C/C++ applications. The patch or modification file does not extend nor provide additional functionality to the Boost libraries.

To access the patch file for building the Boost libraries, go to Boost Library Regression Test Summaries and select `download required Boost modification file` for your compiler release and platform.

You can download the latest Boost libraries at http://www.boost.org/.

For more information on support for libraries, search on the XL C/C++ Compiler support page at http://www.ibm.com/support/entry/portal/overview/software/rational/xl_c~c++_for_linux.

Tools, utilities, and commands

This topic introduces the main tools, utilities, and commands that are included with XL C/C++. It does not contain all compiler tools, utilities, and commands.

Tools

Utilization reporting tool

The utilization reporting tool generates a report describing your organization's utilization of the compiler. These reports help determine whether your organization's use of the compiler matches your compiler license entitlements. The **urt** command contains options that can be used to customize the report. For more information, see Tracking and reporting compiler usage in the *XL C/C++ Compiler Reference*.

Utilities

gxlc and gxlc++ utilities

The **gxlc** and **gxlc++** utilities translate GNU C and GNU C++ invocation commands into corresponding **xlc** and **xlc++** commands before invoking the XL C/C++ compiler. The purpose of these utilities is to minimize the number of changes to makefiles used for existing applications built with the GNU compilers and to facilitate the transition to the XL C/C++ compiler. For more information, see Reusing GNU C/C++ compiler options with gxlc and gxlc++ in the *XL C/C++ Compiler Reference*.

new_install

The **new_install** utility configures IBM XL C/C++ for Linux, V13.1 for use on your system after you install the compiler. Starting from V13.1, you can use the **new_install** utility to facilitate the use of XL C/C++ with IBM Advance Toolchain. For details, see "Using IBM XL C/C++ for Linux, V13.1 with the Advance Toolchain" in *XL C/C++ Compiler Reference*.

xlc_configure

The **xlc_configure** utility creates additional compiler configuration files to contain your own custom sets of compiler option default settings. For more information, see Running the xlc_configure utility directly (for advanced users) in the *XL C/C++ Installation Guide*.

Commands

genhtml command

The **genhtml** command converts an existing XML diagnostic report produced by the **-qlistfmt** option. You can choose to produce XML or HTML diagnostic reports by using the **-qlistfmt** option. The report can help with finding optimization opportunities. For more information about how to use this command, see **genhtml** command in the *XL C/C++ Compiler Reference*.

Profile-directed feedback (PDF) related commands

cleanpdf command

The **cleanpdf** command removes all the PDF files or the specified PDF files from the directory to which profile-directed feedback data is written.

mergepdf command

The **mergepdf** command provides the ability to weigh the importance of two or more PDF records when combining them into a single record. The PDF records must be derived from the same executable.

resetpdf command

The current behavior of the **resetpdf** command is the same as the **cleanpdf** command, and is retained for compatibility with earlier releases on other platforms.

showpdf command

The **showpdf** command displays the following types of profiling information for all the procedures executed in a PDF run (compilation under the **-qpdf1** option):

- Block-counter profiling
- Call-counter profiling
- Value profiling
- Cache-miss profiling, if you specified the **-qpdf1=level=2** option during the **-qpdf1** phase.

You can view the first two types of profiling information in either text or XML format. However, you can view value profiling and cache-miss profiling information only in XML format.

For more information, see -qpdf1, -qpdf2 in the *XL C/C++ Compiler Reference*.

Program optimization

XL C/C++ provides several compiler options that can help you control the optimization and performance of your programs.

With these options, you can perform the following tasks:

- Select different levels of compiler optimizations.
- Control optimizations for loops, floating point, and other types of operations.
- Optimize a program for a particular class of machines or for a very specific machine configuration, depending on where the program will run.

Optimizing transformations can give your application better overall execution performance. XL C/C++ provides a portfolio of optimizing transformations tailored to various supported hardware. These transformations offer the following benefits:

- Reducing the number of instructions executed for critical operations
- Restructuring generated object code to make optimal use of the Power Architecture
- Improving the usage of the memory subsystem
- Exploiting the ability of the architecture to handle large amounts of shared memory parallelization

For more information, see these related topics:

- "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*
- "Optimizing and tuning options" in the *XL C/C++ Compiler Reference*
- "Compiler built-in functions" in the *XL C/C++ Compiler Reference*

64-bit object capability

The XL C/C++ compiler's 64-bit object capability addresses increasing demand for larger storage requirements and greater processing power.

The Linux operating system provides an environment that allows you to develop and execute programs that exploit 64-bit processors through the use of 64-bit address spaces.

To support larger executables that can be fit within a 64-bit address space, a separate 64-bit object format is used. The linker binds these objects to create 64-bit executables. Objects that are bound together must all be of the same object format. The following scenarios are not permitted and will fail to load, execute, or both:

- A 64-bit object or executable that has references to symbols from a 32-bit library or shared library
- A 32-bit object or executable that has references to symbols from a 64-bit library or shared library
- A 64-bit executable that explicitly attempts to load a 32-bit module
- A 32-bit executable that explicitly attempts to load a 64-bit module

XL C/C++ supports 64-bit mode mainly through the use of the **-q64** and **-qarch** compiler options. This combination determines the bit mode and instruction set for the target architecture.

For more information, see "Using 32-bit and 64-bit modes" in the *XL C/C++ Optimization and Programming Guide*

Shared memory parallelization

XL C/C++ supports application development for multiprocessor system architectures.

You can use any of the following methods to develop your parallelized applications with XL C/C++:

- Directive-based shared memory parallelization
- Instructing the compiler to automatically generate shared memory parallelization
- Message passing based shared or distributed memory parallelization (MPI)

For more information, see "Parallelizing your programs" in the *XL C/C++ Optimization and Programming Guide*.

OpenMP directives

OpenMP directives are a set of API-based commands supported by XL C/C++ and many other IBM and non-IBM C, C++, and Fortran compilers.

You can use OpenMP directives to instruct the compiler how to parallelize a particular loop. The existence of the directives in the source removes the need for the compiler to perform any parallel analysis on the parallel code. OpenMP directives require the presence of Pthread libraries to provide the necessary infrastructure for parallelization.

OpenMP directives address three important issues of parallelizing an application:

1. Clauses and directives are available for scoping variables. Frequently, variables should not be shared; that is, each processor should have its own copy of the variable.
2. Work sharing directives specify how the work contained in a parallel region of code should be distributed across the processors.
3. Directives are available to control synchronization between the processors.

As of IBM XL C/C++ for Linux, V13.1, XL C/C++ supports OpenMP API Version 3.1 and selected features of the OpenMP API Version 4.0 specification. For details, see “OpenMP 4.0” on page 14.

For more information about program performance optimization, see:

- "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*
- The OpenMP API specification for parallel programming

Diagnostic listings

The compiler output listings and the XML or HTML reports provide important information to help you develop and debug your applications more efficiently.

Listing information is organized into optional sections that you can include or omit. For more information about the applicable compiler options and the listing itself, see "Compiler messages and listings" in the *XL C/C++ Compiler Reference*.

You can also get the diagnostic information from the compiler in XML or HTML format about some of the optimizations that the compiler performed or missed. You can use this information to reduce programming effort when tuning applications, especially high-performance applications. The report is defined by an XML schema and is easily consumable by tools that you can create to read and analyze the results. For detailed information about this report and how to use it, see "Using reports to diagnose optimization opportunities" in the *XL C/C++ Optimization and Programming Guide*.

Symbolic debugger support

You can instruct XL C/C++ to include debugging information in your compiled objects by using different levels of the **-g** compiler option.

For details, see **-g** in *XL C/C++ Compiler Reference*.

The debugging information can be examined by **gdb** or any other symbolic debugger to help you debug your programs.

Chapter 2. What's new for IBM XL C/C++ for Linux, V13.1

This section describes features and enhancements added to IBM XL C/C++ for Linux, V13.1.

Support for POWER8 processors

XL C/C++ for Linux, V13.1 supports POWER8 processors.

The new features and enhancements introduced in support of the POWER8 processors, fall under the following categories:

- MASS libraries for POWER8 processors
- Compiler options for POWER8 processors
- Built-in functions for POWER8 processors

Mathematical Acceleration Subsystem (MASS) libraries for POWER8 processors

Vector libraries

The vector MASS library **libmassvp8.a** contains vector functions that have been tuned for the POWER8 architecture. The functions can be used in either 32-bit mode or 64-bit mode.

For more information about the vector libraries, see Using the vector libraries in the *XL C/C++ Optimization and Programming Guide*.

SIMD libraries

The MASS SIMD library **libmass_simdp8.a** contains an accelerated set of frequently used math built-in functions that provide improved performance over the corresponding standard system library functions.

For more information about the SIMD libraries, see Using the SIMD libraries in the *XL C/C++ Optimization and Programming Guide*.

Compiler options for POWER8 processors

The **-qarch** compiler option specifies the processor architecture for which code is generated. The **-qtune** compiler option tunes instruction selection, scheduling, and other architecture-dependent performance enhancements to run best on a specific hardware architecture.

The new **-qarch=pwr8** suboption produces object code containing instructions that will run on the POWER8 hardware platforms. With the new **-qtune=pwr8** suboption, optimizations are tuned for the POWER8 hardware platforms.

For more information, see -qarch in the *XL C/C++ Compiler Reference* and -qtune in the *XL C/C++ Compiler Reference*.

Built-in functions for POWER8 processors

New hardware built-in functions are added to support the following POWER8 processor features:

- POWER8 functions for vector processing

© Copyright IBM Corp. 1996, 2014

- POWER8 binary-coded decimal functions
- POWER8 cryptography functions
- POWER8 quad-word arithmetic functions
- POWER8 load-and-reserve/store conditional instructions
- POWER8 cache and data prefetch control functions
- POWER8 transactional memory functions
- POWER8 prefetch functions

For more information about built-in functions provided by XL C/C++, see Compiler built-in functions in the *XL C/C++ Compiler Reference*.

Advance Toolchain 7.0 support

IBM XL C/C++ for Linux, V13.1 fully supports IBM Advance Toolchain 7.0, which is a set of open source development tools and runtime libraries. With IBM Advance Toolchain, you can take advantage of the latest POWER® hardware features on Linux, especially the tuned libraries.

For more information, see "Using IBM XL C/C++ for Linux, V13.1 with the Advance Toolchain" in the *XL C/C++ Compiler Reference*.

C++11 features

In addition to the existing C++11 features, new C++11 features are supported in this release of XL C/C++.

Note: IBM supports selected features of C++11, known as C++0x before its ratification. IBM will continue to develop and implement the features of this standard. The implementation of the language level is based on IBM's interpretation of the standard. Until IBM's implementation of all the C++11 features is complete, including the support of a new C++11 standard library, the implementation may change from release to release. IBM makes no attempt to maintain compatibility, in source, binary, or listings and other compiler interfaces, with earlier releases of IBM's implementation of the new C++11 features.

The following features are introduced in XL C/C++ V13.1:

- Defaulted and deleted functions
- The `nullptr` keyword

The generalized constant expressions feature is enhanced in XL C/C++ V13.1.

You can use the **-qlanglvl=extended0x** option to enable most of the C++ features and all the currently supported C++11 features. For details, see -qlanglvl in the *XL C/C++ Compiler Reference*.

Defaulted and deleted functions

This feature introduces two new forms of function declarations to define explicitly defaulted functions and deleted functions. For the explicitly defaulted functions, the compiler generates the default implementations, which are more efficient than manually programmed implementations. The compiler disables the deleted functions to avoid calling unwanted functions.

You can use the **-qlanglvl=defaultanddelete** option to enable this feature.

For more information, see "Explicitly defaulted functions (C++11)" and "Deleted functions (C++11)" in the *XL C/C++ Language Reference*.

Generalized constant expressions

The generalized constant expressions feature extends the set of expressions permitted within constant expressions. The implementation of this feature in XL C/C++ V12.1 was a partial implementation of what is defined in the C++11 standard. In this release, enhancements are made to support user-defined `constexpr` objects and `constexpr` pointers or references to `constexpr` functions and objects.

You can use the **-qlanglvl=constexpr** option to enable this feature.

For more information, see "Generalized constant expressions (C++11)" in the *XL C/C++ Language Reference*.

The nullptr keyword

This feature introduces `nullptr` as a null pointer constant. The `nullptr` constant can be distinguished from integer 0 for overloaded functions. The constants of 0 and NULL are treated as of the integer type for overloaded functions, whereas `nullptr` can be implicitly converted to only the pointer type, pointer-to-member type, and bool type.

You can use the **-qlanglvl=nullptr** option to enable this feature.

For more information, see langref.pdf#nullpt in the *XL C/C++ Language Reference*.

Related information in the *XL C/C++ Compiler Reference*

-qlanglvl

C11 features

In addition to the existing C11 features, new C11 features are supported in this release of XL C/C++.

Note: IBM supports selected features of C11, known as C1X before its ratification. IBM will continue to develop and implement the features of this standard. The implementation of the language level is based on IBM's interpretation of the standard. Until IBM's implementation of all the C11 features is complete, including the support of a new C11 standard library, the implementation may change from release to release. IBM makes no attempt to maintain compatibility, in source, binary, or listings and other compiler interfaces, with earlier releases of IBM's implementation of the C11 features.

The following features are introduced in IBM XL C/C++ for Linux, V13.1:

- `typedef` redeclaration
- Generic selection

`typedef` redeclaration

Using `typedef` redeclaration, you can redefine a name that is a previous `typedef` name in the same scope to refer to the same type. IBM The XL C compiler supports all types, including a variably modified type. IBM For more

information, see "typedef definitions" in the *XL C/C++ Language Reference*.

Generic selection

Generic selection provides a mechanism to choose an expression according to a given type name at compile time. A common usage is to define type generic macros. For more information, see Generic selection (C11).

OpenMP 4.0

IBM XL C/C++ for Linux, V13.1 partially supports the OpenMP Application Program Interface Version 4.0 specification. The XL C/C++ implementation is based on IBM's interpretation of the OpenMP Application Program Interface Version 4.0.

This version of XL C/C++ supports the following OpenMP 4.0 features:

- `update` and `capture` clauses enhancements
- OMP_DISPLAY_ENV environment variable

`update` and `capture` clauses enhancements

The `update` and `capture` clauses of the `atomic` construct are extended to support more expression forms.

OMP_DISPLAY_ENV environment variable

You can use the OMP_DISPLAY_ENV environment variable to display the values of the internal control variables (ICVs) associated with the environment variables and the build-specific information about the runtime library.

Related information

- "OpenMP environment variables" in the *XL C/C++ Compiler Reference*
- "Pragma directives for parallel processing" in the *XL C/C++ Compiler Reference*
- The OpenMP API specification for parallel programming

Built-in functions

The following major categories of built-in functions are new to this release.

POWER8 built-in functions for vector processing

The following vector built-in functions are added:

- The vector gather-bits-by-bytes doubleword function
  - `vec_gbb`
- The vector count leading zeros function
  - `vec_cntlz`
- The vector population count function
  - `vec_popcnt`
- Extended vector logical operations functions
  - `vec_eqv`
  - `vec_nand`
  - `vec_orc`

- 128-bit integer add subtract functions
  - `vec_add_u128`
  - `vec_sub_u128`
  - `vec_adde_u128`
  - `vec_sube_u128`
  - `vec_addc_u128`
  - `vec_subc_u128`
  - `vec_addec_u128`
  - `vec_subec_u128`
  - `vec_bperm`

The following built-in functions are extended to support doubleword types:

- Vector pack functions
  - `vec_pack`
  - `vec_packs`
  - `vec_packsu`
- Vector unpack functions
  - `vec_unpackh`
  - `vec_unpackl`
- Vector add and subtract functions
  - `vec_add`
  - `vec_sub`
- Vector max and min functions
  - `vec_max`
  - `vec_min`
- Vector shift and rotate functions
  - `vec_rl`
  - `vec_sl`
  - `vec_sr`
  - `vec_sra`
- Vector compare functions
  - `vec_cmpeq`
  - `vec_cmpgt`
  - `vec_cmpge`
  - `vec_cmplt`
  - `vec_cmple`

POWER8 binary-coded decimal built-in functions

The following built-in functions are added to support binary-coded decimal (BCD) arithmetic and comparison:

- BCD add and subtract functions
  - `__bcdadd`
  - `__bcdsub`
- BCD test add and subtract for overflow functions
  - `__bcdadd_ofl`

  - `__bcdsub_ofl`
  - `__bcd_invalid`
- BCD comparison functions
  - `__bcdcmpeq`
  - `__bcdcmpgt`
  - `__bcdcmpge`
  - `__bcdcmplt`
  - `__bcdcmple`
- BCD load and store functions
  - `__vec_ldrmb`
  - `__vec_strmb`

POWER8 cryptography built-in functions

The following built-in functions are provided to perform cryptographic operations:

- Advanced Encryption Standard (AES) functions
  - `__vcipher`
  - `__vcipherlast`
  - `__vncipher`
  - `__vncipherlast`
  - `__vsbox`
- Secure Hash Algorithm (SHA) functions
  - `__vshasigmad`
  - `__vshasigmaw`
- Miscellaneous functions
  - `__vpmsumb`
  - `__vpmsumh`
  - `__vpmsumw`
  - `__vpmsumd`
  - `__vpermxor`

POWER8 non-vector built-in functions

The following built-in functions are added to improve the efficiency of cache:

- `__dcbtna`
- `__icbt`

Load and store built-in functions are extended with the following functions to support more types:

- `__lqarx`
- `__lharx`
- `__lbarx`
- `__stqcx`
- `__sthcx`
- `__stbcx`

POWER8 transactional memory built-in functions

Transactional memory is a model for parallel programming. In this model, you can designate a block of instructions or statements to be treated atomically.

You can use the following built-in functions to mark the beginning or end of transactions, and to diagnose the reasons for failure:

- Transaction begin and end functions
  - `__TM_begin`
  - `__TM_end`
  - `__TM_simple_begin`
- Transaction abort functions
  - `__TM_abort`
  - `__TM_named_abort`
- Transaction inquiry functions
  - `__TM_failure_address`
  - `__TM_failure_code`
  - `__TM_is_conflict`
  - `__TM_is_failure_persistent`
  - `__TM_is_footprint_exceeded`
  - `__TM_is_illegal`
  - `__TM_is_named_user_abort`
  - `__TM_is_nested_too_deep`
  - `__TM_is_user_abort`
  - `__TM_nesting_depth`

POWER8 prefetch built-in functions

The following built-in functions display the problem state control of the Data Stream Control Register (DSCR) in an intuitive, portable, and optimization-friendly way:

- Transient attribute enable functions
  - `__hardware_transient_enable`
  - `__load_transient_enable`
  - `__software_transient_enable`
  - `__store_transient_enable`
- Unit count enable and set functions
  - `__hardware_unit_count_enable`
  - `__software_unit_count_enable`
  - `__set_prefetch_unit_count`
- Prefetch depth functions
  - `__default_prefetch_depth`
  - `__depth_attainment_urgency`
- Load stream enable and disable functions
  - `__load_stream_disable`
  - `__stride_n_stream_enable`
- DSCR functions

- `__prefetch_get_dscr_register`
- `__prefetch_set_dscr_register`

Note: POWER8 built-in functions are valid only when **-qarch** is set to target POWER8 processors.

For more information about built-in functions provided by XL C/C++, see Compiler built-in functions in the *XL C/C++ Compiler Reference*.

Compiler options and pragma directives

This section describes new or changed compiler options and pragma directives.

You can specify compiler options on the command line. You can also modify compiler behavior through pragma directives embedded in your application source files. For detailed descriptions and usage information for XL C/C++ compiler options, see the *XL C/C++ Compiler Reference*.

-qarch The option default is updated to **pwr5**. Suboptions denoting old hardware families are silently upgraded to newer architectures.

The following suboptions are added or updated:

-qarch=pwr7
This suboption produces object code containing instructions that run on the POWER7, POWER7+, or POWER8 hardware platforms.

-qarch=pwr8
This suboption produces object code containing instructions that run on the POWER8 hardware platforms.

-qcheck
The following suboptions are added or updated:

-qcheck=stackclobber
This suboption detects a certain type of stack corruption in your programs.

-qcheck=unset
This suboption checks for automatic variables that are used before they are set at run time.

-qdbgfmt
The following suboption is added:

-qdbgfmt=dwarf
This suboption generates debugging information in DWARF 3 format.

-qdbgfmt=dwarf4
This suboption generates debugging information in DWARF 4 format.

-qhelp This option displays the man page of the compiler.

-qinfo

The compiler does not issue informational messages for the following files:

- Files in the standard search paths for compiler and system header files.
- Files that are ultimately included by the files in the standard search paths for compiler and system header files.

The following suboptions are added or updated:

-qinfo=mt
This suboption notifies you about potential places where synchronization is needed.

-qinfo=unset
This suboption detects automatic variables that are used before they are set, and flags them with informational messages at compile time.

-qlanglvl
The following suboptions are added or updated:

C++11 **-qlanglvl=defaultanddelete**
This suboption enables the defaulted and deleted functions feature, with which you can define explicitly defaulted functions whose implementations are generated by the compiler to achieve higher efficiency. With this feature, you can also define deleted functions whose usages are disabled by the compiler to avoid calling unwanted functions. C++11

C++11 **-qlanglvl=nullptr**
This suboption enables the `nullptr` feature. With this feature, you can initialize a null pointer with the `nullptr` constant. The null pointer can be converted to the pointer type, pointer-to-member type, or bool type. The `nullptr` constant can be distinguished from the integer 0 for overloaded functions. C++11

-qpdf1=unique
This suboption creates a unique PDF file for each process during run time.

-qprefetch=dscr
This suboption helps to improve the runtime performance of your applications. You can specify a value for `dscr` depending on your system architecture.

-qsimd=auto
This suboption controls the autosimdization, which was performed by the deprecated **-qhot=simd** option.

-qstaticlink=xllibs
This suboption links in XL compiler libraries statically.

-qtune The option default is updated.

The following suboptions are added or updated:

-qtune=pwr7
This suboption specifies that optimizations are tuned for the POWER7 or POWER7+ hardware platforms.

-qtune=pwr8
This suboption specifies that optimizations are tuned for the POWER8 hardware platforms.

SMT suboptions
The new **-qtune** simultaneous multithreading (SMT) suboptions allow you to specify a target SMT to direct optimization for best performance in that mode.

-qunroll=*n*
This suboption hints to the compiler to unroll loops by a factor of *n*. If the loop has fewer than *n* iterations, it is fully unrolled.

-qvisibility
This option specifies visibility attributes for entities. Entity visibility attributes describe whether and how entities defined in one module can be referenced or used in other modules. Visibility attributes affect entities with external linkage only, and cannot increase the visibility of other entities.

New or changed pragma directives

#pragma GCC visibility push, #pragma GCC visibility pop
This pair of pragma directives is the pragma equivalent of the **-qvisibility** option. The pragma directives are used to specify visibility attributes for external linkage symbols.

Performance and optimization

Additional features and enhancements assist with performance tuning and application optimization.

IBM

Visibility attributes of entities

Entity visibility attributes describe whether and how an entity that is defined in one module can be referenced or used in other modules. By using the visibility attributes for entities, you can get the following benefits:

- Decrease the size of shared libraries
- Reduce the chance of symbol collision
- Allow more optimization for the compile and link phases
- Improve the efficiency of dynamic linking

For more information, see "Using visibility attributes (IBM extension)" in the *XL C/C++ Optimization and Programming Guide*.

IBM

For more information about performance tuning and program optimization, see "Optimizing your applications" and "Coding your application to improve performance" in the *XL C/C++ Optimization and Programming Guide*.

Chapter 3. Migrating from earlier versions

By migrating to the latest version of the compiler, you are able to take advantage of the new features of the compiler, including features to help boost the performance of the applications being compiled.

A later version of the compiler includes new and enhanced features. These features can provide the following benefits:

- Improved optimization of compiled applications with additional performance benefits
- Support for new language standards to facilitate code portability between multiple operating systems and hardware platforms
- Exploitation of the latest functionality in new hardware and operating systems

Migrating to the latest version of the compiler gives you the ability to exploit the new features with benefits of increased performance optimization, support of new language specifications, and exploitation of new hardware and software environments.

For detailed information about the new features in the current release, see Chapter 2, “What's new for IBM XL C/C++ for Linux, V13.1,” on page 11. The white paper *Upgrading XL C/C++ Compilers* also provides additional information for compiler migration.

In general, a new version of the compiler is compatible with its earlier versions. However, there could be exceptions. For example, different diagnostic messages could be generated. Always back up your source code and other important data before migrating the compiler.

The following sections list the enhancements added to the compiler in earlier versions, which can assist you in migrating to a later version of the compiler.

Enhancements added in Version 12.1

This section describes features and enhancements added to the compiler in Version 12.1.

C++11 features

C++11 is a new C++ programming language standard. Before its ratification, C++11 was called C++0x.In addition to the existing C++11 features, new C++11 features are supported in XL C/C++ V12.1.

Note: IBM supports selected features of C++11, known as C++0x before its ratification. IBM will continue to develop and implement the features of this standard. The implementation of the language level is based on IBM's interpretation of the standard. Until IBM's implementation of all the C++11 features is complete, including the support of a new C++11 standard library, the implementation may change from release to release. IBM makes no attempt to maintain compatibility, in source, binary, or listings and other compiler interfaces, with earlier releases of IBM's implementation of the new C++11 features.

The following features are introduced in XL C/C++, V12.1:

© Copyright IBM Corp. 1996, 2014

- Explicit conversion operators
- Generalized constant expressions
- Reference collapsing
- Right angle brackets
- Rvalue references
- Scoped enumerations
- Trailing return type

You can use the **-qlanglvl=extended0x** option to enable most of the C++ features and all the currently-supported C++11 features. For details, see -qlanglvl in the *XL C/C++ Compiler Reference*.

Explicit conversion operators

The explicit conversion operators feature supports the `explicit` function specifier being applied to the definition of a user-defined conversion function. You can use this feature to inhibit implicit conversions from being applied where they might be unintended, and thus program more robust classes with fewer ambiguity errors.

You can use the **-qlanglvl=explicitconversionoperators** option to enable this feature.

For more information, see "Explicit Conversion Operators (C++11)" in the *XL C/C++ Language Reference*.

Generalized constant expressions

The generalized constant expressions feature extends the set of expressions permitted within constant expressions. A constant expression is one that can be evaluated at compile time.

You can use the **-qlanglvl=constexpr** option to enable this feature.

Note: In XL C/C++ V12.1, this feature is a partial implementation of what is defined in the C++11 standard.

Reference collapsing

With the reference collapsing feature, you can form a reference to a reference type using one of the following contexts:

- A `decltype` specifier
- A `typedef` name
- A template type parameter

You can use the **-qlanglvl=referencecollapsing** option to enable this feature.

For more information, see "Reference collapsing (C++11)" in the *XL C/C++ Language Reference*.

Right angle brackets

In the C++ language, two consecutive closing angle brackets (>) must be separated with a white space, because they are otherwise parsed as the bitwise right-shift

operator (>>). The right angle bracket feature removes the white space requirement for consecutive right angle brackets, thus making programming more convenient.

You can use the **-qlanglvl=rightanglebracket** option to enable this feature.

For more information, see "Class templates (C++ only)" in the *XL C/C++ Language Reference*.

Rvalue references

With the rvalue references feature, you can overload functions based on the value categories of arguments and similarly have lvalueness detected by template argument deduction. You can also have an rvalue bound to an rvalue reference and modify the rvalue through the reference. This enables a programming technique with which you can reuse the resources of expiring objects and therefore improve the performance of your libraries, especially if you use generic code with class types, for example, template data structures. Additionally, the value category can be considered when writing a forwarding function.

You can use the **-qlanglvl=rvaluereferences** option to enable this feature.

For more information, see "Using rvalue references (C++11)" in the *XL C/C++ Optimization and Programming Guide*.

Scoped enumerations

With the scoped enumeration feature, you can get the following benefits:

- The ability to declare a scoped enumeration type, whose enumerators are declared in the scope of the enumeration.
- The ability to declare an enumeration without providing the enumerators. The declaration of an enumeration without providing the enumerators is referred to as forward declaration.
- The ability to specify explicitly the underlying type of an enumeration.
- Improved type safety with no conversions from the value of an enumerator (or an object of an enumeration type) to an integer.

You can use the **-qlanglvl=scopedenum** option to enable this feature.

For more information, see "Enumeration" in the *XL C/C++ Language Reference*.

Trailing return type

The trailing return type feature is useful when declaring the following types of templates and functions:

- Function templates or member functions of class templates with return types that depend on the types of the function arguments
- Functions or member functions of classes with complicated return types
- Perfect forwarding functions

You can use the **-qlanglvl=autotypededuction** option to enable this feature.

For more information, see "Trailing return type (C++11)" in the *XL C/C++ Language Reference*.

Related information in the *XL C/C++ Compiler Reference*

-qlanglvl

C11 features

XL C/C++ V12.1 introduces support for selected features of C11, which is a new C programming language standard.

Note: IBM supports selected features of C11, known as C1X before its ratification. IBM will continue to develop and implement the features of this standard. The implementation of the language level is based on IBM's interpretation of the standard. Until IBM's implementation of all the C11 features is complete, including the support of a new C11 standard library, the implementation may change from release to release. IBM makes no attempt to maintain compatibility, in source, binary, or listings and other compiler interfaces, with earlier releases of IBM's implementation of the C11 features.

The following C11 features are introduced in XL C/C++, V12.1:

- Anonymous structures
- Complex type initialization
- New language level - **extc1x**
- The `_Noreturn` function specifier
- Static assertions

Anonymous structures

This feature enables the declaration of anonymous structures under the **extc1x** language level. For more information, see "Anonymous structures" in the *XL C/C++ Language Reference*.

Complex type initialization

Macros CMPLX, CMPLXF, and CMPLXL are defined inside the standard header file complex.h to enable the initialization of complex types under the **extc1x** language level. For more information, see "Initialization of complex types (C11)" in the *XL C/C++ Language Reference*.

New language level - extc1x

A new suboption has been added to the **-qlanglvl** option in this release. When you compile with the C compiler, you can use **-qlanglvl=extc1x** to enable C11 features that are currently supported by XL C/C++. Certain C11 features are also available when you compile with the C++ compiler. For further information, see the sections that describe individual features.

The `_Noreturn` function specifier

The `_Noreturn` function specifier declares that a function does not return to its caller. You can define your own functions that do not return using this function specifier. The compiler can produce better code by ignoring what would happen if the function returns. For more information, see "The _Noreturn function specifier" in the *XL C/C++ Language Reference*.

Static assertions

The addition of static assertions to the C language has the following benefits:

- Libraries can detect common usage errors at compile time.
- Implementations of the C Standard Library can detect and diagnose common usage errors, improving usability.

You can declare static assertions to check important program invariants at compile time.

For more information, see "_Static_assert declaration (C11)" in the *XL C/C++ Language Reference*.

OpenMP 3.1

XL C/C++ V12.1 supports the OpenMP Application Program Interface Version 3.1 specification. The XL C/C++ implementation is based on IBM's interpretation of the OpenMP Application Program Interface Version 3.1.

OpenMP 3.1 includes the following updates to OpenMP 3.0:

- Adds the `final` and `mergeable` clauses to the `task` construct to support optimization.
- Adds the `taskyield` construct to allow users to specify where in the program can perform task switching.
- Adds the `omp_in_final` runtime library function to support specialization of final task regions.
- Extends the `atomic` construct to include `read`, `write`, and `capture` forms; adds the `update` clause to apply the existing form of the `atomic` construct.
- Adds two reduction operators: `min` and `max`.
- Allows const-qualified types to be specified on the `firstprivate` clause.
- Adds the `OMP_PROC_BIND` environment variable to control whether OpenMP threads are allowed to move between processors.
- Extends the `OMP_NUM_THREADS` environment variable to specify the number of threads to use for nested parallel regions.

Related information

- "OpenMP environment variables" in the *XL C/C++ Compiler Reference*
- "Pragma directives for parallel processing" in the *XL C/C++ Compiler Reference*
- www.openmp.org

Performance and optimization

Additional features and enhancements in XL C/C++ V12.1 assist with performance tuning and application optimization.

Reports about compiler optimizations

There are a number of enhancements to the listing reports to give you more information about how the compiler optimized your code. You can use this information to get further benefits from the optimization capabilities of the compiler. For more details about these enhanced reports, see “Diagnostic reports” on page 26.

For additional information about performance tuning and program optimization, see "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*.

Diagnostic reports

The new diagnostic reports added in XL C/C++ V12.1 can help you identify opportunities to improve the performance of your code.

Compiler reports in HTML format

It is now possible to get information in XML or HTML format about the optimizations that the compiler was able to perform and also which optimization opportunities were missed. This information can be used to reduce programming effort for tuning applications, especially high-performance applications.

The **-qlistfmt** option and its associated suboptions can be used to generate the XML or HTML report. By default, this option now generates all the available content if you do not specify the type of content.

To view the HTML version of an XML report that has been already generated, you can now use the **genhtml** tool. For more information about how to use this tool, see the **genhtml** command in the *XL C/C++ Compiler Reference*.

For detailed information about this report and how to use it, see "Using reports to diagnose optimization opportunities" in the *XL C/C++ Optimization and Programming Guide*.

Enhancements to profiling reports

New sections have been added to your listing file to help you analyze your programs. When using the **-qreport** option with the **-qpdf2** option, you can get the following sections added to the listing file in the section entitled `PDF Report`:

Relevance of profiling data
: This section shows the relevance of the profiling data to the source code during the **-qpdf1** phase. The relevance is indicated by a number in the range of 0 - 100. The larger the number is, the more relevant the profiling data is to the source code, and the more performance gain can be achieved by using the profiling data.

Missing profiling data
: This section might include a warning message about missing profiling data. The warning message is issued for each function for which the compiler does not find profiling data.

Outdated profiling data
: This section might include a warning message about outdated profiling data. The compiler issues this warning message for each function that is modified after the **-qpdf1** phase. The warning message is also issued when the optimization level changes from the **-qpdf1** phase to the **-qpdf2** phase.

For detailed information about profile-directed feedback, see "Using profile-directed feedback" in the *XL C/C++ Optimization and Programming Guide*.

For additional information about the listing files, see "Compiler listings" in the *XL C/C++ Compiler Reference*.

Enhancements to showpdf reports

In addition to block-counter and call-counter profiling information currently provided, you can also use the **showpdf** utility to view cache-miss profiling and value profiling information. Value profiling and cache-miss profiling information can be displayed only in XML format. However, all the other types of profiling information can be displayed in either text or XML format. In this release, the profile-directed feedback (PDF) information is saved in two files. One is a PDF map file that is generated during the **-qpdf1** phase, and the other is a PDF file that is generated during the execution of the resulting application. You can run the **showpdf** utility to display the PDF information contained in these two files. For more information, see "Viewing profiling information with showpdf" in the *XL C/C++ Optimization and Programming Guide*.

New and enhanced diagnostic options

The entries in the following table describe new or changed compiler options and directives that give you control over compiler listings.

The information presented here is a brief overview. For detailed information about these and other performance-related compiler options, see "Listings, messages and compiler information" in the *XL C/C++ Compiler Reference*.

Table 4. Listings-related compiler options and directives

Option/directive	Description
-qlistfmt	The **-qlistfmt** option has been enhanced to generate HTML reports as well as XML reports, containing information about optimizations performed by the compiler and missed optimization opportunities. The default behavior of this option has changed. Now, if you do not specify a particular type of content, the option generates all the available content, rather than generating none.

Built-in functions

This section describes the major categories of built-in functions that are new for V12.1.

GCC atomic memory access built-in functions (IBM extension)

New XL C/C++ built-in functions for atomic memory access, whose behavior corresponds to that provided by GNU Compiler Collection (GCC), are added in this release. In a program with multiple threads, you can use these functions to atomically and safely modify data in one thread without interference from another thread.

For more information about built-in functions provided by XL C/C++, see Compiler built-in functions in the *XL C/C++ Compiler Reference*.

Compiler options and pragma directives

This section describes new or changed compiler options and pragma directives in V12.1.

You can specify compiler options on the command line. You can also modify compiler behavior through pragma directives embedded in your application source files. See the *XL C/C++ Compiler Reference* for detailed descriptions and usage information for these and other compiler options.

New or changed compiler options

-g
The **-g** option is extended to have new different levels to improve the debugging of optimized programs.

-qhaltonmsg
The **-qhaltonmsg** option, previously supported only by the C++ compiler, is now supported by XL C. It stops compilation before producing any object files, executable files, or assembler source files if a specified error message is generated. The negative form **-qnohaltonmsg** has also been added.

-qinclude

The negative form **-qnoinclude** is added to ignore the previously specified **-qinclude** option.

-qinfo
-qinfo=all now enables all diagnostic messages for all groups except **als** and **ppt**

-qinitauto
The **-qinitauto** option is enhanced to be able to perform word initialization for automatic variables.

-qkeyword
C++11 The new suboption **-q[no]keyword=constexpr** enables or disables the `constexpr` keyword.

-qlanglvl
The following suboptions are added or updated:

C++11 **-qlanglvl=autotypededuction**
This suboption can now enable the trailing return type feature in addition to the auto type deduction feature.

C++ **-qlanglvl=c1xnoreturn**
This suboption enables support for the `_Noreturn` function specifier.

C++ **-qlanglvl=complexinit**
This suboption controls whether to enable the initialization of complex types.

C++ IBM **-qlanglvl=compatrvaluebinding**
This suboption instructs the compiler to allow a non-const lvalue reference to bind to an rvalue of a user-defined type where an initializer is not required.

C++11 **-qlanglvl=constexpr**
This suboption enables the generalized constant expressions feature, which extends the expressions permitted within constant expressions.

Note: In XL C/C++ V12.1, this feature is a partial implementation of what is defined in the C++11 standard.

C++11 **-qlanglvl=explicitconversionoperators**
This suboption enables the explicit conversion operators feature,

which allows you to inhibit unintended implicit conversions through the user-defined conversion function.

C11 **-qlanglvl=extc1x**
This suboption enables all the currently supported C11 features and other implementation-specific language extensions.

C++11 **-qlanglvl=referencecollapsing**
This suboption enables the reference collapsing feature, with which you can form a reference to a reference type using a `decltype` specifier, a `typedef` name, or a template type parameter.

C++11 **-qlanglvl=rightanglebracket**
This suboption enables the right angle bracket feature, which removes the white space requirement for consecutive right angle brackets.

C++11 **-qlanglvl=rvaluereferences**
This suboption enables the rvalue references feature.

C++11 **-qlanglvl=scopedenum**
This suboption enables the scoped enumeration feature, with which you can declare a scoped enumeration type or an enumeration without providing the enumerators.

C++ IBM **-qlanglvl=tempsaslocals**
This suboption extends the lifetime of temporaries to reduce migration difficulty.

IBM **-qlanglvl=textafterendif**
This suboption suppresses the warning message that is emitted when you are porting code from a compiler that allows extra text after `#endif` or `#else` to IBM XL C/C++ compiler.

For more information about the new C++11 features, see "C++11 features" on page 21.

For more information about the C11 features, see "C11 features" on page 24.

-qlistfmt
The **-qlistfmt** option is enhanced to generate HTML reports as well as XML reports, containing information about optimizations performed by the compiler and missed optimization opportunities.

The default behavior of **-qlistfmt** has changed. In this release, if you do not specify a particular type of content, the option generates all the available content, rather than generating none.

-qoptfile
The new option **-qoptfile** specifies a file containing a list of additional command line options to be used for the compilation.

-qpic **-qpic=large** now enables large TOC access and prevents TOC overflow conditions when the Table of Contents is larger than 64 Kb.

-qshowpdf
The default value is changed from **-qnoshowpdf** to **-qshowpdf**.

New or changed pragma directives

#pragma ibm independent_loop
The **independent_loop** pragma is added. It explicitly states that the iterations of the chosen loop are independent, and that the iterations can be executed in parallel.

#pragma ibm iterations
The **iterations** pragma is added. It specifies the approximate number of loop iterations for the chosen loop.

#pragma ibm max_iterations
The **max_iterations** pragma is added. It specifies the approximate maximum number of loop iterations for the chosen loop.

#pragma ibm min_iterations
The **min_iterations** pragma is added. It specifies the approximate minimum number of loop iterations for the chosen loop.

#pragma simd_level
The **simd_level** pragma is added. It controls the compiler code generation of vector instructions for individual loops.

Enhancements added in Version 11.1

This section describes features and enhancements added to the compiler in Version 11.1.

Support for POWER7 processors

XL C/C++ for Linux, V11.1 supports POWER7 processors.

The new features and enhancements introduced in support for the POWER7 processors, fall under the following four categories:

- Vector scalar extension data types and built-in functions
- MASS libraries for POWER7 processors
- Built-in functions for POWER7 processors
- Compiler options for POWER7 processors

Vector scalar extension data types and built-in functions

This release of the compiler supports the Vector Scalar eXtension (VSX) instruction set in the POWER7 processors. New data types and built-in functions are introduced to support the VSX instructions. With the VSX built-in functions and the original Vector Multimedia eXtension (VMX) built-in functions, you can efficiently manipulate vector operations in your application.

For more information about the VSX data types and built-in functions, see Vector types in the *XL C/C++ Language Reference* and Vector built-in functions in the *XL C/C++ Compiler Reference*.

Mathematical Acceleration Subsystem (MASS) libraries for POWER7 processors

Vector libraries

The vector MASS library **libmassvp7.a** contains vector functions that have been tuned for the POWER7 architecture. The functions can be used in either 32-bit mode or 64-bit mode.

Functions supporting previous Power® processors, either single-precision or double-precision, are included for POWER7 processors.

The following new functions are added, in both single-precision and double-precision function groups:

- exp2
- exp2m1
- log21p
- log2

For more information about the vector libraries, see Using the vector libraries in the *XL C/C++ Optimization and Programming Guide*.

SIMD libraries

The MASS SIMD library **libmass_simdp7.a** contains an accelerated set of frequently used math built-in functions that provide improved performance over the corresponding standard system library functions.

For more information about the SIMD libraries, see Using the SIMD library for POWER7 in the *XL C/C++ Optimization and Programming Guide*.

POWER7 hardware built-ins

New hardware built-in functions are added to support the following POWER7 processor features:

- New POWER7 prefetch extensions and cache control
- New POWER7 hardware instructions

For more information, see "Built-in functions" on page 42.

New compiler options for POWER7 processors

New arch and tune compiler options

The **-qarch** compiler option specifies the processor architecture for which code is generated. The **-qtune** compiler option tunes instruction selection, scheduling, and other architecture-dependent performance enhancements to run best on a specific hardware architecture.

-qarch=pwr7 produces object code containing instructions that will run on the POWER7 hardware platforms. With **-qtune=pwr7**, optimizations are tuned for the POWER7 hardware platforms.

For more information, see -qarch in the *XL C/C++ Compiler Reference* and -qtune in the *XL C/C++ Compiler Reference*.

C++11 features

XL C/C++, V11.1 introduces support for selected features of C++11, which is a new C++ programming language standard.

Note: IBM supports selected features of C++11, known as C++0x before its ratification. IBM will continue to develop and implement the features of this standard. The implementation of the language level is based on IBM's interpretation of the standard. Until IBM's implementation of all the C++11 features is complete, including the support of a new C++11 standard library, the implementation may change from release to release. IBM makes no attempt to

maintain compatibility, in source, binary, or listings and other compiler interfaces, with earlier releases of IBM's implementation of the new C++11 features.

The following features are introduced in XL C/C++, V11.1:

- Auto type deduction
- C99 `long long`
- C99 preprocessor features adopted in C++11
- Decltype
- Delegating constructors
- Explicit instantiation declarations
- Extended friend declarations
- Inline namespace definitions
- Static assertion
- Variadic templates

You can use the **-qlanglvl=extended0x** option to enable most of the C++ features and all the currently-supported C++11 features. For details, see -qlanglvl in the *XL C/C++ Compiler Reference*.

Auto type deduction

With the auto type deduction feature, you no longer need to specify a type while declaring a variable. This is because auto type deduction delegates the task of deducting the type of an auto variable to the compiler from the type of its initializer expression.

You can use the **-qlanglvl=autotypedededuction** option to enable this feature.

For more information, see "The auto type specifier (C++11)" in the *XL C/C++ Language Reference*.

C99 `long long`

The C++ compiler can use the C99 `long long` feature, which improves source compatibility between the C and C++ languages.

You can use the **-qlanglvl=c99longlong** option to enable the C99 `long long` feature.

IBM After this feature is enabled, if a decimal integer literal that does not have a suffix containing `u` or `U` cannot be represented by the `long long int` type, you can decide whether to use the `unsigned long long int` type to represent the literal or not by specifying the `-qlanglvl=[no]extendedintegersafe` option.

For more information, see "Integer literals" in the *XL C/C++ Language Reference*.

C99 preprocessor features adopted in C++11

With several C99 preprocessor features adopted in C++11, C and C++ compilers provide a more common preprocessor interface, which can ease porting C source files to the C++ compiler, eliminate semantic differences between the C and C++ preprocessors, and avoid preprocessor compatibility issues or diverging preprocessor behaviors.

You can use the **-qlanglvl=c99preprocessor** option to enable this feature.

For more information, see "C99 preprocessor features adopted in C++11)" in the *XL C/C++ Language Reference*.

Decltype

With the decltype feature, you can get a type that is based on the resultant type of a possibly type-dependent expression.

You can use the **-qlanglvl=decltype** option to enable this feature.

For more information, see "The decltype(expression) type specifier (C++11)" in the *XL C/C++ Language Reference*.

Delegating constructors

With the delegating constructors feature, you can concentrate common initializations in one constructor, which makes programs more readable and maintainable.

You can use the **-qlanglvl=delegatingctors** option to enable this feature.

For more information, see "Delegating constructors (C++11)" in the *XL C/C++ Language Reference*.

Explicit instantiation declarations

With the explicit instantiation declarations feature, you can suppress the implicit instantiation of a template specialization or its members.

You can use the individual suboption **-qlanglvl=externtemplate** or the group options **-qlanglvl=extended** or **-qlanglvl=extended0x** to enable this feature.

For more information, see "Explicit instantiation (C++ only)" in the *XL C/C++ Language Reference*.

Extended friend declarations

The extended friend declarations feature relaxes the syntax rules governing friend declarations as follows:

- Template parameters, `typedef` names, and basic types can be declared as friends.
- The class-key in the context for friend declarations is no longer necessary in C++11.

You can use the **-qlanglvl=extendedfriend** option to enable this feature.

For more information, see "Friends (C++ only)" in the *XL C/C++ Language Reference*.

Inline namespace definitions

Inline namespace definitions are namespace definitions with an initial `inline` keyword. You can define or specialize the members of an inline namespace as if they belong to the enclosing namespace that contains the inline namespace.

You can use the **-qlanglvl=inlinenamespace** option to enable this feature.

For more information, see "Inline namespace definitions (C++11)" in the *XL C/C++ Language Reference*.

Static assertion

The static assertion feature provides you with the following benefits:

- Libraries can detect common usage errors at compile time.
- Implementations of the C++ Standard Library can detect and diagnose common usage errors, thus improving usability.

You can use a static_assert declaration to check important program invariants at compile time.

You can use the **-qlanglvl=static_assert** option to enable this feature.

For more information, see "static_assert declaration (C++11)" in the *XL C/C++ Language Reference*.

Variadic templates

With the variadic templates feature, you can define class or function templates that have any number (including zero) of parameters.

You can use the **-qlanglvl=variadic[templates]** option to enable this feature.

For more information, see "Variadic templates (C++11)" in the *XL C/C++ Language Reference*.

Related information in the *XL C/C++ Compiler Reference*

-qlanglvl

Performance and optimization

Additional features and enhancements assist with performance tuning and application optimization.

Enhancements to -qpdf

The use of the **-qpdf** option consists of two steps. First, compile your program with the **-qpdf1** option and run it with a typical set of data to generate the profiling data. Second, compile your program again with the **-qpdf2** option to optimize the program based on the profiling data.

In previous releases, if you modify the source files and compile them with the **-qpdf2** option, the compilation stops with an error. As of XL C/C++ for Linux, V11.1, the compiler issues a list of warnings but the compilation does not stop. This allows you to continue using the profiling data after modifying the source files.

Some new suboptions are added to the **-qpdf** option. You can use these new suboptions to get more control over performance improvements and extend **-qpdf** to support multiple-pass profiling, cache-miss profiling, and extended value profiling.

The new **-qpdf** suboptions are:

level Supports multiple-pass profiling, single-pass profiling, cache-miss profiling, value profiling, block-counter profiling, and call-counter profiling. You can compile your program with **-qpdf1=level=0|1|2** to specify the type of profiling information to be generated by the resulting application.

exename
Specifies the name of the generated PDF file according to the output file name specified by the **-o** option.

defname
Reverts the PDF file to its default file name.

For detailed information about these suboptions, see -qpdf1, -qpdf2 in the *XL C/C++ Compiler Reference*.

Reports about compiler optimizations

There are a number of enhancements to the listing reports to give you more information about how the compiler optimizes your code. You can use this information to get further benefits from the optimization capabilities of the compiler. For more details about these enhanced reports, see "New diagnostic reports" on page 36.

Performance-related compiler options and directives

The entries in the following table describe new or changed compiler options and directives.

Information presented here is a brief overview. For detailed information about these options, directives, and other performance-related compiler options, see "Optimization and tuning options" in the *XL C/C++ Compiler Reference*.

Table 5. Performance-related compiler options and directives

-qinline=level=*number*	A new option is added to -qinline to provide guidance to the compiler about the relative value of inlining in relation to the default value of 5.*number* is a range of integer values 0 - 10 that indicates the level of inlining you want to use. For details, see -qinline in the *XL C/C++ Compiler Reference*.
-qpdf	**-qpdf** provides suboptions to give you more control flexibility in controlling different PDF optimizations. For more information, see the -qpdf1, -qpdf2 section in the *XL C/C++ Compiler Reference*.
-qprefetch	A new enhancement is added to **-qprefetch** for inserting prefetch instructions automatically where there are opportunities to improve code performance: `-qprefetch=assistthread`. For details, see -qprefetch in the *XL C/C++ Compiler Reference*.

For additional information about performance tuning and program optimization, see "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*.

New diagnostic reports

The new diagnostic reports can help you identify opportunities to improve the performance of your code.

Compiler reports in XML format

It is now possible to get information in XML format about the optimizations that the compiler was able to perform and also which optimization opportunities were missed. This information can be used to reduce programming effort for tuning applications, especially high-performance applications.

The information from the compiler is produced in XML 1.0 format. The report is defined by an XML schema and is easily consumable by tools that you can create to read and analyze the results. A stylesheet, `xlstyle.xsl`, is provided to render the report into a human readable format that can be read by anyone with a browser which supports XSLT.

In this release, the following four optimization categories are available in the report:

- Inlining
- Loop transformations
- Data reorganizations
- Profile-directed feedback information

The new **-qlistfmt** option and its associated suboptions can be used to generate the new XML 1.0 report.

For detailed information about this report and how to use it, see "Using reports to diagnose optimization opportunities" in the *XL C/C++ Optimization and Programming Guide*.

Enhancements to profiling reports

New sections have been added to your listing file to help you analyze your programs. When using the **-qreport** option with the **-qpdf2** option, you can get the following sections added to the listing file in the section entitled `PDF Report`:

Loop iteration count
: The most frequent loop iteration count and the average iteration count, for a given set of input data, is calculated for most loops in a program. This information is only available when the program is compiled at optimization level -O5.

Block and call count
: This section of the report covers the call structure of the program and the respective execution count for each called function. It also includes block information for each function. For non-user defined functions, only execution count is given. The total block and call coverage, and a list of the user functions ordered by decreasing execution count are printed in the end of this report section. In addition, the block count information is printed at the beginning of each block of the pseudo-code in the listing files.

Cache miss
: This section of the report is printed in a single table. It reports the number

of cache misses for certain functions, with additional information about the functions such as: cache level, cache miss ratio, line number, file name, and memory reference.

Note: You must use the **-qpdf1=level=2** option to get this report. You can also select the level of cache to profile using the **PDF_PM_EVENT** environment variable during run time.

For detailed information about profile-directed feedback, see "Using profile-directed feedback" in the *XL C/C++ Optimization and Programming Guide*.

For additional information about the listing files, see "Compiler listings" in the *XL C/C++ Compiler Reference*.

Report of data reorganization

The compiler can generate the following information in the listing files:

- Data reorganizations (a summary of how program variable data gets reorganized by the compiler)
- The location of data prefetch instructions inserted by the compiler

To generate data reorganization information, specify the optimization level **-qipa=level=2** or **-O5** together with **-qreport**. The data reorganization messages for program variable data are added to the data reorganization section of the listing file with the label `DATA REORGANIZATION SECTION` during the IPA link pass. Reorganizations include:

- array splitting
- array transposing
- memory allocation merging
- array interleaving
- array coalescing

To generate information about data prefetch insertion locations, use the optimization level of **-qhot**, or any other option that implies **-qhot** together with **-qreport**. This information appears in the `LOOP TRANSFORMATION SECTION` of the listing file.

Additional loop analysis

A new suboption has been added to **-qhot** to add more aggressive loop analysis. **-qhot=level=2** together with **-qsmp** and **-qreport** add information about loop nests on which the aggressive loop analysis was performed to the `LOOP TRANSFORMATION SECTION` of the listing file. This information can also appear in the XML listing file created with the **-qlistfmt** option.

New and enhanced diagnostic options

The entries in the following table describe new or changed compiler options and directives that give you control over compiler listings.

Information presented here is a brief overview. For detailed information about these and other performance-related compiler options, see "Listings, messages and compiler information" in the *XL C/C++ Compiler Reference*.

Table 6. Listings-related compiler options and directives

Option/directive	Description
-qlistfmt	Generates a report in an XML 1.0 format containing information about optimizations performed by the compiler and missed optimization opportunities. The report contains information about inlining, loop transformations, data reorganization and profile-directed feedback.
-qreport	The listing now contains a `PDF report` section when used with **-qpdf2**. Another new section in the listing files is a `DATA REORGANIZATION` section when used with **-qipa=level=2** or **-O5**.
-qskipsrc	Determines whether the source statements skipped by the compiler are shown in the `SOURCE` section of the listing file.

Utilization tracking and reporting tool

The utilization tracking and reporting feature is a lightweight and simple mechanism for tracking the compiler utilization within your organization. It is disabled by default. You can use this feature to detect whether your organization's use of the compiler exceeds your compiler license entitlements.

When utilization tracking is enabled, each invocation of the compiler is recorded in a compiler utilization file. You can run the utilization reporting tool to generate a report from one or more of these files to get a picture of the overall usage of the compiler within your organization. The **urt** command can be used to control how the report is generated. In particular, the report indicates the number of concurrent users using the compiler.

The utilization tracking and reporting feature is easy to set up and manage, and utilization tracking does not impact the usage or performance of the compiler.

For detailed information about the utilization tracking and reporting feature, see "Tracking and reporting compiler usage" in the *XL C/C++ Compiler Reference*.

New or changed compiler options and directives

This section describes new and changed compiler options and directives in XL C/C++, V11.1.

You can specify compiler options on the command line. You can also modify compiler behavior through pragma directives embedded in your application source files. See the *XL C/C++ Compiler Reference* for detailed descriptions and usage information for these and other compiler options.

Table 7. New or changed compiler options and directives

Option or directive	Description
-qarch	A new suboption has been added to **-qarch**, specifying **-qarch=pwr7** produces object code that contains instructions that run on the POWER7 hardware platforms.
-qassert	**-qassert** is a new option for XL C/C++. It is used to provide information about the characteristics of the files that can help to fine-tune optimizations.

Table 7. New or changed compiler options and directives (continued)

Option or directive	Description
-qfunctrace	Traces the entry and exit points of functions in a compilation unit or only for a specific list of functions.
-qhot	A new suboption has been added for **-qhot**. The **-qhot** compiler option is a powerful alternative to hand tuning that provides opportunities to optimize loops and array language. The **-qhot=fastmath** option enables the replacement of math routines with available math routines from the XLOPT library only if **-qstrict=nolibrary** is enabled. **-qhot=nofastmath** disables the replacement of math routines by the XLOPT library. -qhot=fastmath is enabled by default if -qhot is specified regardless of the hot level.
-qinline	Attempts to inline functions instead of generating calls to those functions, for improved performance.
-qkeepinlines	A new suboption **exports** has been added to the **-qkeepinlines** option. You can use **-qkeepinlines=exports** to make sure that the compiler keeps the list of symbols and their definitions from the shared object file compiled with an earlier version of the compiler.

Table 7. New or changed compiler options and directives (continued)

Option or directive	Description
-qlanglvl	C++11 New suboptions have been added to **-qlanglvl**: • **-qlanglvl=autotypededuction**: Controls whether the auto type deduction feature is enabled. This feature can be used to delegate the task of type deduction of an auto variable to the compiler from the type of its initializer expression. • **-qlanglvl=c99longlong**: Controls whether the C99 `long long` feature is enabled. This feature improves source compatibility between the C and C++ languages. • **-qlanglvl=c99preprocessor**: Controls whether the C99 preprocessor features adopted in C++11 are enabled. This feature can be used to provide a more common preprocessor interface for C and C++ compilers. • **-qlanglvl=decltype**: Controls whether the decltype feature is enabled. This feature can be used to get a type that is based on the resultant type of a possibly type-dependent expression. • **-qlanglvl=delegatingctors**: Controls whether the delegating constructors feature is enabled. This feature can be used to concentrate common initializations in one constructor. • **-qlanglvl=extendedfriend**: Controls whether the extended friend declarations feature is enabled. This feature can be used to accept additional forms of non-function friend declarations. • IBM **-qlanglvl=extendedintegersafe**: Controls whether or not `unsigned long long int` can be used as the type for decimal integer literals that do not have a suffix containing `u` or `U` and cannot be represented by the `long long int` type. This option takes effect only when the **-qlanglvl=c99longlong** option is specified. • **-qlanglvl=externtemplate**: Controls whether the explicit instantiation declarations feature is enabled. This feature can be used to suppress the implicit instantiation of a template specialization or its members. • **-qlanglvl=inlinenamespace**: Controls whether the inline namespace definitions feature is enabled. This feature can be used to define and specialize members of an inline namespace as if they were also members of the enclosing namespace. • **-qlanglvl=static_assert**: Controls whether the static assertions feature is enabled. This feature can be used to produce compile-time assertions for which a severe error message is issued on failure. • **-qlanglvl=variadic[templates]**: Controls whether the variadic templates feature is enabled. This feature can be used to define class or function templates that have any number (including zero) of parameters.

Table 7. New or changed compiler options and directives (continued)

Option or directive	Description
-qlibmpi	Tunes code based on the known behavior of the Message Passing Interface (MPI) functions.
-qlistfmt	Generates a report in an XML 1.0 format containing information about some optimizations performed by the compiler and some missed optimization opportunities for inlining, loop transformations, profile-directed feedback, and data reorganization.
-qpdf1,-qpdf2	New suboptions have been added to **-qpdf1,-qpdf2**.
-qprefetch	A new suboption has been added to **-qprefetch**. When you work with applications that generate a high cache-miss rate, you can use **-qprefetch=assistthread** to exploit assist threads for data prefetching.
-qrestrict (C only)	You can use **-qrestrict** to indicate to the compiler that no other pointer can access the same memory that has been addressed by function parameter pointers.
-qsaveopt|-qnosaveopt	The existing **-qsaveopt** option is enhanced to also include the user's configuration file name and the options specified in the configuration files.
-qstackprotect	Protects your applications against malicious code or programming errors that overwrite or corrupt the stack.
-qstaticlink	Controls how shared and nonshared runtime libraries are linked into an application.
-qstrict	A new suboption has been added to the **-qstrict** option to allow more control over optimizations and transformations that violate strict program semantics. **-qstrict=vectorprecision** disables vectorization in loops where it might produce different results in vectorized iterations than in nonvectorized ones.
-qtune	A new suboption has been added to **-qtune**. If you specify **-qtune=pwr7**, optimizations are tuned for the POWER7 hardware platforms.

Table 8. Deprecated directives and options

Option or directive	Description
-Q	This option is deprecated and replaced with **-qinline**.
-qenablevmx	This option is deprecated and replaced with the **-qsimd=auto** option.
-qhot=simd | nosimd	**-qhot=simd | nosimd** are deprecated and might be removed in a future release. You can use **-qsimd**.
-qinfo=private	**-qinfo=private** is deprecated and replaced with **-qreport**.
-qinfo=reduction	**-qinfo=reduction** is deprecated and replaced with **-qreport**.
-qipa=inline | noinline	**-qipa=inline | noinline** are deprecated and might be removed in a future release. You can use **-qinline**.

Table 8. Deprecated directives and options (continued)

Option or directive	Description
-qipa=clonearch | noclonearch	**-qipa=clonearch | noclonearch** is no longer supported. You can use **-qtune=balanced**.
-qipa=clonearch | noclonearch	**-qipa=cloneproc | nocloneproc** is no longer supported. You can use **-qtune=balanced**.

Built-in functions

This section lists built-in functions that are new for XL C/C++, V11.1.

For more information about built-in functions provided by XL C/C++, see Compiler built-in functions in the *XL C/C++ Compiler Reference*.

VSX built-in functions

Vector Scalar eXtension (VSX) is newly added for POWER7 processors.

For more information about VSX built-in functions, see Vector built-in functions.

POWER7 prefetch extensions and cache control

The POWER7 processor has cache control and stream prefetch extensions that support store stream prefetch and prefetch depth control. XL C/C++ provides the following new built-in functions to provide direct programmer access to these instructions:

- `__protected_stream_stride`
- `__transient_protected_stream_count_depth`
- `__unlimited_protected_stream_depth`
- `__transient_unlimited_protected_stream_depth`
- `__partial_dcbt`
- `__dcbtt`
- `__dcbtstt`
- `__dcbflp`

The compiler can insert the built-in functions automatically when it optimizes the code. You can disable automatic use of these instructions with **-qnoprefetch**.

For more information about the directives, see built-in functions in the *XL C/C++ Compiler Reference*.

POWER7 hardware built-in functions

New XL C/C++ built-in functions corresponding to each new POWER7 hardware instruction are added in this release. With these functions, you can directly manipulate specific hardware instructions in your code, which can improve the performance of your application.

- `__bpermd`
- `__cbcdtd`
- `__cdtbcd`
- `__load8r`

- `__store8r`
- `__divde`
- `__divdeu`
- `__cmpb`
- `__divwe`
- `__divweu`
- `__addg6s`

Conversion functions

These new functions convert between Declets and Binary Coded Decimal.

- `__cbcdtd`
- `__cdtbcd`

Comparison functions

This new function compares bytes.

- `__cmpb`

Decimal floating-point functions

This new function adds and generates sixes.

- `__addg6s`

Enhancements added in Version 10.1

This section describes features and enhancements added to the compiler in Version 10.1.

C++11 features

XL C/C++ V10.1 introduces support for a new version of the standard for the C++ programming language - specifically C++11, also known as C++0x before its ratification. The standard was not officially adopted at the time of the release but we began to support some of its features.

Note: IBM supports selected features of C++11, known as C++0x before its ratification. IBM will continue to develop and implement the features of this standard. The implementation of the language level is based on IBM's interpretation of the standard. Until IBM's implementation of all the C++11 features is complete, including the support of a new C++11 standard library, the implementation may change from release to release. IBM makes no attempt to maintain compatibility, in source, binary, or listings and other compiler interfaces, with earlier releases of IBM's implementation of the new C++11 features.

Specifically, in this release:

- we add a new language level
- we introduce new integer promotion rules for arithmetic conversions with `long long` data types
- the C++ preprocessor now supports C99 features

New language level - extended0x

The default **-qlanglvl** compiler option remains **extended** when invoking the C++ compiler.

A new suboption has been added to the **-qlanglvl** option in this release. **-qlanglvl=extended0x** is used to allow users to try out early implementations of any features of C++11 that are currently supported by XL C/C++.

C99 long long under C++

With this release of XL C/C++, V10.1, compiler behavior changes when performing certain arithmetic operations with integral literal data types. Specifically, the integer promotion rules have changed.

Previously, in C++ (and as an extension to C89), when compiling with **-qlonglong**, an unsuffixed integral literal would be promoted to the first type in this list into which it fitted:

```
int
long int
unsigned long int
long long int
unsigned long long
```

Starting with this release and when compiling with **-qlanglvl=extended0x**, the compiler now promotes unsuffixed integral literal to the first type in this list into which it fits:

```
int
long int
long long int
unsigned long long
```

Note: Like our implementation of the C99 Standard in the C compiler, C++ will allow promotions from `long long` to `unsigned long long` if a value cannot fit into a `long long` type, but can fit in an `unsigned long long`. In this case, a message will be generated.

The macro `__C99_LLONG` has been added for compatibility with C99. This macro is defined to `1` with **-qlanglvl=extended0x** and is otherwise undefined.

For more information, see "Integral and floating-point promotions" in the *XL C/C++ Language Reference*.

Preprocessor changes

The following changes to the C++ preprocessor make it easier to port code from C to C++:

- Regular string literals can now be concatenated with wide-string literals.
- The `#line <integer>` preprocessor directive has a larger upper limit. It has been increased from 32767 to 2147483647 for C++ .
- C++ now supports `_Pragma` operator.
- These macros now apply to C++ as well as C:

- __C99_MACRO_WITH_VA_ARGS (also available with **-qlanglvl=extended**)
- __C99_MAX_LINE_NUMBER (also available with **-qlanglvl=extended**)
- __C99_PRAGMA_OPERATOR
- __C99_MIXED_STRING_CONCAT

Note: Except as noted, these C++ preprocessor changes are only available when compiling with **-qlanglvl=extended0x**.

For additional information about the language standards supported by XL C/C++, see "Language levels and extensions" in the *XL C/C++ Language Reference*.

Other XL C/C++ language-related updates

This section describes the language-related changes introduced in Version 10.1.

Vector data types

Vector data types can now use some of the operators that can be used with base data types such as:.

- unary operators
- binary operators
- relational operators

OpenMP 3.0

IBM XL C/C++ for Linux, V10.1, supports the OpenMP API Version 3.0 specification. The XL C/C++ implementation is based on IBM's interpretation of the OpenMP Application Program Interface Draft 3.0 Public Comment.

The main differences between Version 2.5 and Version 3.0 are:

- Addition of task level parallelization. The new OpenMP constructs `TASK` and `TASKWAIT` give users the ability to parallelize irregular algorithms, such as pointer chasing or recursive algorithms for which the existing OpenMP constructs were not adequate.
- `for` loops can now contain *var* values of `unsigned int` and `pointer` type as well as `signed int`.
- Stack size control. You can now control the size of the stack for threads created by the OMP runtime library using the new environment variable `OMP_STACKSIZE`.
- Users can give hints to the expected behavior of waiting threads using new environment variables `OMP_WAIT_POLICY` and `OMP_SET_POLICY`.
- Storage reuse. Some restrictions on the `PRIVATE` clause have been removed. A list item that appears in the `reduction` clause of a parallel construct can now also appear in a `private` clause on a work-sharing construct.
- Scheduling. A new `SCHEDULE` attribute, `auto`, allows the compiler and runtime system to control scheduling.
- Consecutive loop constructs with `STATIC` schedule can now use `nowait`.
- Nesting support - a `COLLAPSE` clause has been added to the `DO`, `FOR`, `PARALLEL FOR`, and `PARALLEL DO` directives to allow parallelization of perfect loop nests. This means that multiple loops in a nest can be parallelized.
- `THREADPRIVATE` directives can now apply to variables at class scope in addition to file and block scope.
- Parallelization of iterator loops of canonical form including those with random access iterators.

For more information, see:

- "Using OpenMP directives" in the *XL C/C++ Optimization and Programming Guide*
- www.openmp.org

Performance and optimization

XL C/C++, V10.1 includes features and enhancements to assist with performance tuning and optimization of your applications.

Enhancements to -qstrict

Many suboptions have been added to the **-qstrict** option to allow more fine-grained control over optimizations and transformations that violate strict program semantics. In previous releases, the **-qstrict** option disabled all transformations that violate strict program semantics. This is still the behavior if you use **-qstrict** without suboptions. Likewise, in previous releases **-qnostrict** allowed transformations that could change program semantics. Because a higher level of optimizations might require relaxing strict program semantics, the addition of the suboptions relaxes selected rules to get specific benefits of faster code without turning off all semantic verifications.

You can use 16 new suboptions separately or use a suboption group. Here is a list of suboption groups:

all Disables all semantics-changing transformations, including those controlled by the other suboptions.

ieeefp
Controls whether individual operations conform to IEEE 754 semantics.

order Controls whether individual operations can be reordered in a way that violate program language semantics.

precision
Controls optimizations and transformations that can affect the precision of program results.

exceptions
Controls optimizations and transformations that can affect the runtime exceptions generated by the program.

For detailed information about these suboptions, see "-qstrict" in the *XL C/C++ Compiler Reference*.

Performance-related compiler options and directives

The entries in the following table describe new or changed compiler options and directives.

Information presented here is a brief overview. For detailed information about these and other performance-related compiler options, see "Optimization and tuning options" in the *XL C/C++ Compiler Reference*.

Table 9. Performance-related compiler options and directives

Option/directive	Description
-qstrict	Many new suboptions have been added to give you more control over the relaxation of program semantic rules in order to gain some performance benefits.

Table 9. Performance-related compiler options and directives (continued)

Option/directive	Description
-qfloat	Some **-qfloat** suboptions are affected by the new suboptions for **-qstrict**.
-qreport	The listing now contains information about how many streams are created for each loop and which loops cannot be SIMD vectorized due to non-stride-one references. You can use this information to improve the performance of your applications.
-qsmp	When **-qsmp=omp** is in effect, the additional functionality of OpenMP API 3.0 is now available. For more information, see "OpenMP 3.0" on page 45.

For additional information about performance tuning and program optimization, see "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*.

Compiler options and pragma directives

This section describes new and changed compiler options and directives in XL C/C++, V10.1.

You can specify compiler options on the command line. You can also modify compiler behavior through pragma directives embedded in your application source files. See the *XL C/C++ Compiler Reference* for detailed descriptions and usage information for these and other compiler options.

Table 10. New or changed compiler options and directives

Option or directive	Description
-qstrict	Many suboptions have been added to the **-qstrict** option to allow more control over optimizations and transformations that violate strict program semantics. See "Performance and optimization" on page 34 for more information.
-qshowmacros	When used in conjunction with the **-E** option, the **-qshowmacros** option replaces preprocessed output with macro definitions. There are suboptions provided to control the emissions of predefined and user-defined macros more precisely.
-qreport	When used together with compiler options that enable automatic parallelization or vectorization, the **-qreport** option now reports the number of streams in a loop and produces information when loops cannot be SIMD vectorized due to non-stride-one references.
-qsmp	When **-qsmp=omp** is in effect, the additional functionality of OpenMP API 3.0 is now available. For more information, see "OpenMP 3.0" on page 45.
-qtimestamps	This option can be used to remove timestamps from generated binaries.
-qtls	The thread local storage support has been enhanced to include `__attribute__((tls-model("string")))` where *string* is one of `local-exec`, `initial-exec`, `local-dynamic`, or `global-dynamic`.

Table 10. New or changed compiler options and directives (continued)

Option or directive	Description
-qinfo	The suboptions `als` and `noals` have been added to the **qinfo** option to report (or not report) possible violations of the ANSI aliasing rule.

Predefined macros

This section provides information about the predefined macros added in the release.

Four new macros were added in XL C/C++ V10.1:

_ILP32 __ILP32__
: Defined to 1 only when the compilation is for a target where `long int`, `int` and pointers all use 32 bits. Otherwise it is not defined.

_LP64 __LP64__
: Defined to 1 only when the compilation is for a target where `long int` and pointers both use 64 bits and `int` uses 32 bits. Otherwise it is not defined

The compiler no longer supports the `__C99_COMPLEX_HEADER__` macro.

For a complete list of the predefined macros for XL C/C++, see "Compiler predefined macros" in the *XL C/C++ Compiler Reference*.

Chapter 4. Setting up and customizing XL C/C++

For complete prerequisite and installation information for XL C/C++, see "Before installing XL C/C++" in the *XL C/C++ Installation Guide*.

Using custom compiler configuration files

You can customize compiler settings and options by modifying the default configuration file or creating your own configuration file.

You have the following options to customize compiler settings:

- The XL C/C++ compiler installation process creates a default compiler configuration file. You can directly modify this configuration file to add default options for specific needs. However, if you later apply updates to the compiler, you must reapply all of your modifications to the newly installed configuration file.
- You can create your own custom configuration file that either overrides or complements the default configuration file. The compiler can recognize and resolve compiler settings that you specify in your custom configuration files with compiler settings that are specified in the default configuration file. Compiler updates that might later affect settings in the default configuration file does not affect the settings in your custom configuration files.

For more information, see "Using custom compiler configuration files" in the *XL C/C++ Compiler Reference*.

Configuring compiler utilization tracking and reporting

In addition to the compiler configuration file, there is a separate configuration file for the utilization tracking and reporting feature. Utilization tracking is disabled by default, but you can enable it by modifying an entry in this configuration file. Various other aspects of utilization tracking can also be configured using this file.

Although the compiler configuration file is separate from the utilization tracking configuration file, it contains an entry that specifies the location of the utilization tracking configuration file so that the compiler can find this file.

For more information about how to configure the utilization tracking and reporting feature, see Tracking and reporting compiler usage in the *XL C/C++ Compiler Reference*.

© Copyright IBM Corp. 1996, 2014

Chapter 5. Developing applications with XL C/C++

C/C++ application development consists of repeating cycles of editing, compiling, linking, and running. By default, compiling and linking are combined into a single step.

Notes:

1. Before you can use the compiler, you must first ensure that XL C/C++ is properly installed and configured. For more information, see the *XL C/C++ Installation Guide*.
2. To learn about writing C/C++ programs, refer to the *XL C/C++ Language Reference*.

The compiler phases

A typical compiler invocation executes some or all of these activities in sequence. For link time optimizations, some activities are executed more than once during a compilation. As each compilation component runs, the results are sent to the next step in the sequence.

1. Preprocessing of source files
2. Compilation, which may consist of the following phases, depending on what compiler options are specified:
   a. Front-end parsing and semantic analysis
   b. High-level optimization
   c. Low-level optimization
   d. Register allocation
   e. Final assembly
3. Assemble the assembly (**.s**) files, and the unpreprocessed assembler (**.S**) files after they are preprocessed
4. Object linking to create an executable application

To see the compiler step through these phases, specify the **-v** compiler option when you compile your application. To see the amount of time the compiler spends in each phase, specify **-qphsinfo**.

Editing C/C++ source files

To create C/C++ source programs, you can use any text editor available to your system.

Source programs must be saved using a recognized file name suffix. See "XL C/C++ input and output files" on page 54 for a list of suffixes recognized by XL C/C++.

For a C or C++ source program to be a valid program, it must conform to the language definitions specified in the *XL C/C++ Language Reference*.

© Copyright IBM Corp. 1996, 2014

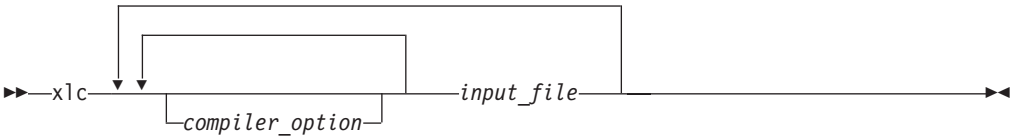
Compiling with XL C/C++

XL C/C++ is a command-line compiler. Invocation commands and options can be selected according to the needs of a particular C/C++ application.

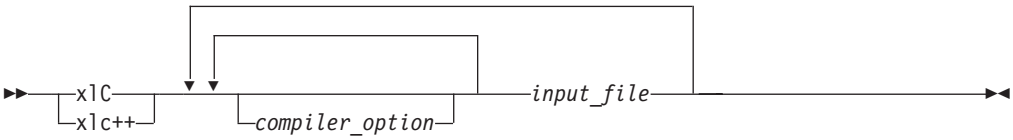
Invoking the compiler

The compiler invocation commands perform all necessary steps to compile C or C++ source files, assemble any **.s** and **.S** files, and link the object files and libraries into an executable program.

To compile a C source program, use the following basic invocation syntax:



To compile a C++ source program, use the following basic invocation syntax:



For most applications, compile with **xlc**, **xlc++**, or a threadsafe counterpart. You can use **xlc++** to compile either C or C++ program source, but compiling C++ files with **xlc** might result in link or runtime errors because libraries required for C++ code are not specified when the linker is called by the C compiler.

More invocation commands are available to meet specialized compilation needs, primarily to provide explicit compilation support for different levels and extensions of the C or C++ language. See "Invoking the compiler" in the *XL C/C++ Compiler Reference* for more information about compiler invocation commands available to you, including special invocations that are intended to assist developers in migrating from a GNU compilation environment to XL C/C++.

Compiling parallelized XL C/C++ applications

XL C/C++ provides threadsafe compiler invocation commands to compile parallelized applications for use in multiprocessor environments.

These invocations are similar to their corresponding base compiler invocations, except that they link and bind compiled objects to threadsafe components and libraries. The generic XL C/C++ threadsafe compiler invocations are as follows:

- xlC_r
- xlc++_r
- xlc_r

XL C/C++ provides additional threadsafe invocations to meet specific compilation requirements. For more information, see "Invoking the compiler" in the *XL C/C++ Compiler Reference*.

Note: Using any of these commands alone does not imply parallelization. For the compiler to recognize OpenMP directives and activate parallelization, you must also specify **-qsmp** compiler option. In turn, you should specify the **-qsmp** option only in conjunction with one of these threadsafe invocation commands. When you specify **-qsmp**, the driver links in the libraries specified on the smp libraries line in the active stanza of the configuration file.

For more information on parallelized applications, see "Parallelizing your programs" in the *XL C/C++ Optimization and Programming Guide*.

Specifying compiler options

Compiler options perform a variety of functions, such as setting compiler characteristics, describing the object code to be produced, controlling the diagnostic messages emitted, and performing some preprocessor functions.

You can specify compiler options in one or any combination of the following ways:

- On the command-line with command-line compiler options
- In your source code using directive statements
- In a makefile
- In the stanzas found in a compiler configuration file

You can also pass compiler options to the linker, assembler, and preprocessor.

For more information about compiler options and their usage, see "Compiler options reference" in the *XL C/C++ Compiler Reference* .

Priority sequence of compiler options

It is possible for option conflicts and incompatibilities to occur when multiple compiler options are specified. To resolve these conflicts in a consistent fashion, the compiler usually applies the following general priority sequence to most options:

1. Directive statements in your source file *override* command-line settings
2. Command-line compiler option settings *override* configuration file settings
3. Configuration file settings *override* default settings

Generally, if the same compiler option is specified more than once on a command-line when invoking the compiler, the last option specified prevails.

Note: Some compiler options, such as the **-I** option, do not follow the priority sequence described above. The compiler searches any directories specified with **-I** in the xlc.cfg file before it searches the directories specified with **-I** on the command-line. The **-I** option is cumulative rather than preemptive. Other options with cumulative behavior are **-R** and **-l** (lowercase L).

Reusing GNU C/C++ compiler options with gxlc and gxlc++

XL C/C++ includes various features to help you transition from GNU C/C++ compilers to XL C/C++, including the gxlc and gxlc++ commands.

Each of the gxlc and gxlc++ utilities accepts GNU C or C++ compiler options and translates them into comparable XL C/C++ options. Both utilities use the XL C/C++ options to create an **xlc** or **xlc++** invocation command, which is then used to invoke the compiler. These utilities are provided to help you reuse makefiles

created for applications previously developed with GNU C/C++. However, to fully exploit the capabilities of XL C/C++, you can use the XL C/C++ invocation commands and their associated options.

The actions of gxlc and gxlc++ are controlled by the `gxlc.cfg` configuration file. The GNU C/C++ options that have an XL C/C++ counterpart are shown in this file. Not every GNU option has a corresponding XL C/C++ option. gxlc and gxlc++ return warnings for input options that were not translated.

The gxlc and gxlc++ option mappings are modifiable. For information on using the gxlc or gxlc++ configuration file, see "Reusing GNU C/C++ compiler options with gxlc and gxlc++ " in the *XL C/C++ Compiler Reference* .

XL C/C++ input and output files

These file types are recognized by XL C/C++.

For detailed information about these and additional file types used by the compiler, see "Types of input files" in the *XL C/C++ Compiler Reference* and "Types of output files" in the *XL C/C++ Compiler Reference*.

Table 11. Input file types

Filename extension	Description
.c	C source files
.C, .cc, .cp, .cpp, .cxx, .c++	C++ source files
.i	Preprocessed source files
.o	Object files
.s	Assembler files
.S	Unpreprocessed assembler files
.so	Shared object or library files

Table 12. Output file types

Filename extension	Description
a.out	Default name for executable file created by the compiler
.d	Make dependency file
.i	Preprocessed source files
.lst	Listing files
.o	Object files
.s	Assembler files
.so	Shared object or library files

Linking your compiled applications with XL C/C++

By default, you do not need to do anything special to link an XL C/C++ program. The compiler invocation commands automatically call the linker to produce an executable output file.

For example, you can use the following command to compile `file1.C` and `file3.C` to produce the object files `file1.o` and `file3.o`. All object files, including `file2.o`, are submitted to the linker to produce one executable.

```
xlc++ file1.C file2.o file3.C
```

Compiling and linking in separate steps

To produce object files that can be linked later, use the **-c** option.

```
xlc++ -c file1.C              # Produce one object file (file1.o)
xlc++ -c file2.C file3.C      # Or multiple object files (file1.o, file3.o)
xlc++ file1.o file2.o file3.o  # Link object files with default libraries
```

For more information about compiling and linking your programs, see:

- "Linking" in the *XL C/C++ Compiler Reference*
- "Constructing a library" in the *XL C/C++ Optimization and Programming Guide*

Dynamic and static linking

XL C/C++ allows your programs to take advantage of the operating system facilities for both dynamic and static linking.

Dynamic linking means that the code for some external routines is located and loaded when the program is first run. When you compile a program that uses shared libraries, the shared libraries are dynamically linked to your program by default. Dynamically linked programs take up less disk space and less virtual memory if more than one program uses the routines in the shared libraries. During linking, they do not require any special precautions to avoid naming conflicts with library routines. They are designed to perform better than statically linked programs if several programs use the same shared routines at the same time. By using dynamic linking, you can upgrade the routines in the shared libraries without relinking.

Because this form of linking is the default, you need no additional options to turn it on.

Static linking means that the code for all routines called by your program becomes part of the executable file.

Statically linked programs can be moved to run on systems without the XL C/C++ runtime libraries. They might perform better than dynamically linked programs if they make many calls to library routines or call many small routines. They do require some precautions in choosing names for data objects and routines in the program if you want to avoid naming conflicts with library routines. They also might not work if you compile them on one level of the operating system and run them on a different level of the operating system.

Running your compiled application

After a program is compiled and linked, you can run the generated executable file on the command line.

The default file name for the program executable file produced by the XL C/C++ compiler is **a.out**. You can select a different name with the **-o** compiler option.

You should avoid giving your program executable file the same name as system or shell commands, such as `test` or `cp`, as you could accidentally execute the wrong command. If you do decide to name your program executable file with the same

name as a system or shell command, you should execute your program by specifying the path name to the directory in which your executable file resides, such as `./test`.

To run a program, enter the name of the program executable file together with any run time arguments on the command line.

Canceling execution

To suspend a running program, press the **Ctrl+Z** key while the program is in the foreground. Use the **fg** command to resume running.

To cancel a running program, press the **Ctrl+C** key while the program is in the foreground.

Setting runtime options

You can use environment variable settings to control certain runtime options and behaviors of applications created with the XL C/C++ compiler. Some environment variables do not control actual runtime behavior, but they can have an impact on how your applications run.

For more information on environment variables and how they can affect your applications at run time, see the *XL C/C++ Installation Guide*.

Running compiled applications on other systems

If you want to run an application developed with the XL C/C++ compiler on another system that does not have the compiler installed, you need to install a runtime environment on that system or link your application statically.

You can obtain the latest XL C/C++ Runtime Environment PTF images, together with licensing and usage information, from the XL C/C++ for Linux support page.

XL C/C++ compiler diagnostic aids

XL C/C++ issues diagnostic messages when it encounters problems compiling your application. You can use these messages and other information provided in compiler output listings to help identify and correct such problems.

For more information about listing, diagnostics, and related compiler options that can help you resolve problems with your application, see the following topics in the *XL C/C++ Compiler Reference*:

- "Compiler messages and listings"
- "Error checking and debugging options"
- "Listings, messages, and compiler information options"

Debugging compiled applications

You can use a symbolic debugger to debug applications compiled with XL C/C++.

At compile time, you can use the **-g** or **-qlinedebug** option to instruct the XL C/C++ compiler to include debugging information in compiled output. For **-g**, you can also use different levels to balance between debug capability and compiler optimization. For more information about the debugging options, see "Error checking and debugging" in the *XL C/C++ Compiler Reference*.

You can then use **gdb** or any other symbolic debugger to step through and inspect the behavior of your compiled application.

Optimized applications pose special challenges when debugging. When debugging highly optimized applications, you should consider using the **-qoptdebug** compiler option. For more information about optimizing your code, see "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*.

Determining which level of XL C/C++ is being used

To display the version and release level of XL C/C++ that you are using, invoke the compiler with the **-qversion** compiler option.

For example, to obtain detailed version information, enter the following command:

```
xlc++-qversion=verbose
```

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 1996, 2014

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd. Laboratory
8200 Warden Avenue
Markham, Ontario L6G 1C7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1998, 2014.

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Index

Special characters

.a files 54
.c and .C files 54
.i files 54
.lst files 54
.mod files 54
.o files 54
.s files 54
.S files 54
.so files 54

Numerics

64-bit environment 7

A

archive files 54
assembler
  source (.s) files 54
  source (.S) files 54

B

basic example, described ix
built-in functions 14, 27, 42

C

C++11 31
  auto type deduction 31
  C99 long long 31
  C99 preprocessor features adopted in C++11 31
  decltype 31
  defaulted and deleted functions 12
  delegating constructors 31
  explicit conversion operators 21
  explicit instantiation declarations 31
  extended friend declarations 31
  inline namespace definitions 31
  reference collapsing 21
  rvalue references 21
  scoped enumerations 21
  static assertion 31
  trailing return type 21
  variadic templates 31
C11 13
C1X 13
  _Static_assert 24
code optimization 7
compilation
  sequence of activities 51
compiler
  controlling behavior of 53
  invoking 52
  running 52
compiler directives
  new or changed 18, 28, 38
compiler options
  conflicts and incompatibilities 53
  new or changed 18, 28, 38
  specification methods 53
compiling
  SMP programs 52
customization
  for compatibility with GNU 3

D

debugger support 56
  output listings 56
  symbolic 9
debugging 56
debugging compiled applications 56
debugging information, generating 56
dynamic linking 55

E

editing source files 51
executable files 54
executing a program 55
executing the linker 55

F

files
  editing source 51
  input 54
  output 54

G

GNU
  compatibility with 3

I

input files 54
invocation commands 52
invoking a program 55
invoking the compiler 52

L

language standards 3
language support 3
level of XL C/C++, determining 57
libraries 54
linking
  dynamic 55
  static 55
linking process 54
listings 54

M

migrate 21
migration 21
  source code 53
mod files 54
multiprocessor systems 8, 45

O

object files 54
  creating 55
  linking 55
OMP directives 45
OpenMP 8
optimization 20
  programs 7
output files 54

P

parallelization 8, 45
performance 20
  optimizing transformations 7
problem determination 56
programs
  running 55

R

running the compiler 52
runtime
  libraries 54
runtime environment 56
runtime options 56

S

shared memory parallelization 8, 45
shared object files 54
SMP
  programs, compiling 52
SMP programs 8
source files 54
source-level debugging support 9
static linking 55
symbolic debugger support 9

T

tools 5
  cleanpdf utility 6
  configuration file utility 6
  gxlc and gxlc++ utilities 5
  mergepdf utility 6
  new install configuration utility 6
  new_install utility 6
  resetpdf utility 6
  showpdf utility 6

© Copyright IBM Corp. 1996, 2014

tools *(continued)*
xlc_configure 6

U

utilities 5
cleanpdf 6
gxlc and gxlc++ 5
mergepdf 6
new_install 6
resetpdf 6
showpdf 6
xlc_configure 6

X

xlc_configure 6
xlc.cfg file 53

Product Number: 5765-J08; 5725-C73

Printed in USA

SC27-4247-00