IBM XL C/C++ for Linux, V13.1.5

Getting Started with XL C/C++ for Little Endian Distributions

Version 13.1.5

GI13-2875-04

IBM XL C/C++ for Linux, V13.1.5

Getting Started with XL C/C++ for Little Endian Distributions

Version 13.1.5

GI13-2875-04

Note

Before using this information and the product it supports, read the information in "Notices" on page 39.

First edition

This edition applies to IBM XL C/C++ for Linux, V13.1.5 (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.

Contents

About this document

This document contains overview and basic usage information for the IBM® XL C/C++ for Linux, V13.1.5 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, 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 about 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” on page vi.

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

While this document covers information such as 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*.
- Compiler options: see the *XL C/C++ Compiler Reference* for detailed information about the syntax and usage of compiler options.
- The C or C++ programming language: see the *XL C/C++ Language Reference* for information about the syntax, semantics, and IBM implementation of the C or C++ IBM extension features. See C/C++ standards for the details of standard features.
- Programming topics: see the *XL C/C++ Optimization and Programming Guide* for detailed information about developing applications with XL C/C++, with a focus on program portability and optimization.

How this document is organized

Chapter 1, “Introducing XL C/C++,” on page 1 contains information about the features of the XL C/C++ compiler at a high level.

Chapter 2, “What's new for IBM XL C/C++ for Linux, V13.1.5,” on page 9 describes features and enhancements added to IBM XL C/C++ for Linux, V13.1.5.

Chapter 3, “Migration of your applications,” on page 15 lists important considerations when you migrate your applications that were compiled with other versions of XL C/C++ or with GCC.

Chapter 4, "Enhancements added in earlier releases," on page 19 describes enhancements added in earlier releases. These enhancements also apply to the current release.

Chapter 5, "Setting up and customizing XL C/C++," on page 29 describes how to set up and customize the compiler according to your own requirements.

Chapter 6, "Developing applications with XL C/C++," on page 31 consists of repeating cycles of editing, compiling, linking, and running.

Conventions

Typographical conventions

The following table shows the typographical conventions used in the IBM XL C/C++ for Linux, V13.1.5 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 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.

Table 2. Qualifying elements (continued)

Qualifier/Icon	**Meaning**
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.
C11 begins ▶ C11 C11 ◀ C11 ends	The text describes a feature that is introduced into standard C as part of C11.
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.
C++14 begins ▶ C++14 C++14 ◀ C++14 ends	The text describes a feature that is introduced into standard C++ as part of C++14.
IBM extension begins ▶ IBM IBM ◀ IBM extension ends	The text describes a feature that is an IBM extension to the standard language specifications.
GPU begins ▶ GPU GPU ◀ GPU ends	The text describes the information that is relevant to offloading computations to the NVIDIA GPUs.

Syntax diagrams

Throughout this information, diagrams illustrate XL C/C++ syntax. This section helps 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.

Example of a syntax statement

EXAMPLE *char_constant* {*a*|*b*}[*c*|*d*]*e*[,*e*]... *name_list*{*name_list*}...

The following list explains the syntax statement:
- Enter the keyword EXAMPLE.
- Enter a value for *char_constant*.
- Enter a value for *a* or *b*, but not for both.
- Optionally, enter a value for *c* or *d*.
- Enter at least one value for *e*. If you enter more than one value, you must put a comma between each.

- Optionally, enter the value of at least one *name* for *name_list*. If you enter more than one value, you must put a comma between each *name*.

Note: The same example is used in both the syntax-statement and syntax-diagram representations.

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:

- Quick Start Guide

  The Quick Start Guide (`quickstart.pdf`) is intended to get you started with IBM XL C/C++ for Linux, V13.1.5. It is located by default in the XL C/C++ directory and in the `\quickstart` directory of the installation DVD.
- 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 and subdirectories of the installation DVD.
- 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, V13.1.5 Installation Guide*.
- Online product documentation

  The fully searchable HTML-based documentation is viewable in IBM Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSXVZZ_13.1.5/com.ibm.compilers.linux.doc/welcome.html.
- PDF documents

  PDF documents are 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, V13.1.5 Installation Guide, GC27-6540-04	`install.pdf`	Contains information for installing XL C/C++ and configuring your environment for basic compilation and program execution.

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

Document title	PDF file name	Description
Getting Started with IBM XL C/C++ for Linux, V13.1.5, GI13-2875-04	`getstart.pdf`	Contains an introduction to the XL C/C++ product, with information about setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
IBM XL C/C++ for Linux, V13.1.5 Compiler Reference, SC27-6570-04	`compiler.pdf`	Contains information about the various compiler options, pragmas, macros, environment variables, and built-in functions.
IBM XL C/C++ for Linux, V13.1.5 Language Reference, SC27-6550-04	`langref.pdf`	Contains information about language extensions for portability and conformance to nonproprietary standards.
IBM XL C/C++ for Linux, V13.1.5 Optimization and Programming Guide, SC27-6560-04	`proguide.pdf`	Contains information about advanced programming topics, such as application porting, interlanguage calls with Fortran code, library development, application optimization, 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, and other articles, is available on the web at http://www.ibm.com/support/docview.wss?uid=swg27036675.

For more information about the compiler, see the XL compiler on Power® community at http://ibm.biz/xl-power-compilers.

Standards and specifications

XL C/C++ is designed to support the following standards and specifications. You can refer to these standards and specifications 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*.
- *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 *C++03*.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2011*, also known as *C++11*.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2014*, also known as *C++14* (Partial support).

- *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 3.1* (full support), *OpenMP Application Program Interface Version 4.0 (partial support)*, and *OpenMP Application Program Interface Version 4.5 (partial support)*, available at http://www.openmp.org

Other IBM information

- *ESSL product documentation* available at http://www.ibm.com/support/knowledgecenter/SSFHY8/essl_welcome.html?lang=en

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/product/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 an email to compinfo@ca.ibm.com.

For the latest information about XL C/C++, visit the product information site at http://ibm.biz/xlcpp-linux.

How to send your comments

Your feedback is important in helping us 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 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, V13.1.5 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, V13.1.5 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®/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 code portability across multiple operating systems and hardware platforms.

Starting from IBM XL C/C++ for Linux, V13.1.1, XL C/C++ combines the Clang front end infrastructure with the advanced optimization technology in the IBM compiler back end. Clang is a component of the LLVM open source compiler and toolchain project and provides the C and C++ language family front end for LLVM. For additional information about Clang, see the LLVM web site at: http://clang.llvm.org/

Operating system and hardware support

This section describes the operating systems and hardware that IBM XL C/C++ for Linux, V13.1.5 supports.

IBM XL C/C++ for Linux, V13.1.5, for little endian distributions supports the following operating systems:

- Ubuntu Server 14.04
- Ubuntu Server 14.10
- Ubuntu Server 16.04
- SUSE Linux Enterprise Server 12 (SLES 12)
- SUSE Linux Enterprise Server 12 Service Pack 1 (SLES 12 SP1)
- Red Hat Enterprise Linux 7.1 (RHEL 7.1)
- Red Hat Enterprise Linux 7.2 (RHEL 7.2)
- Red Hat Enterprise Linux 7.3 (RHEL 7.3)
- Community Enterprise Operating System 7 (CentOS 7)

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

The compiler, its libraries, and its generated object programs run on any IBM Power Systems™ server supported by your operating system distribution 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.

Compiler invocation commands

XL C/C++ provides several commands to invoke the compiler, for example, **xlC**, **xlc++**, and **xlc**. Compiler invocation commands are provided to support most standardized C/C++ language levels and many popular language extensions.

All the invocation commands can be used to link programs that use multithreading. The **_r** versions of invocation commands are for backward compatibility only.

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 tuning 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 29.

Language standard compliance

This topic describes the C/C++ programming language specifications that IBM XL C/C++ for Linux, V13.1.5 supports.

C language specifications
- 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 14882:2014 (referred to as C++14)
- ISO/IEC 14882:2011 (referred to as C++11)
- ISO/IEC 14882:2003 (referred to as C++03)
- ISO/IEC 14882:1998, the first official specification of the C++ language (referred to as C++98)

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

IBM XL C/C++ for Linux, V13.1.5 provides a greater level of GNU source compatibility than previous releases. It supports the use of **gcc** and **g++** compiler options and therefore the **gxlc** and **gxlc++** invocation commands are not required or included.

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, you must install the prerequisite GCC compiler before you install XL C/C++.

Note: Some additional noteworthy points about this relationship are as follows:
- 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.
- Code compiled with XL C/C++ can be debugged with 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 **-std(-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.

Related information

-std (-qlanglvl)

Libraries

XL C/C++ includes a runtime environment that contains 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 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, 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*.

XL C++ Runtime Library

The following library is also shipped with XL C/C++:

- XL C++ Runtime Library contains support routines needed by the compiler.

Support for Boost libraries

IBM XL C/C++ for Linux, V13.1.5 provides partial support for the Boost V1.59.0 libraries. A patch file is available that modifies the Boost V1.59.0 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.

To build and install Boost libraries with the Boost build tool **b2/bjam** using IBM XL C/C++ for Linux V13.1.3 or later releases, you must specify the toolset as **xlcpp**. For more information about the **xlcpp** toolset, see Building Boost libraries using the xlcpp toolset for IBM compilers on Linux.

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

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

Utilities and commands

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

Utilities

install
The **install** utility installs and configures IBM XL C/C++ for Linux, V13.1.5 for use on your system.

xlc_configure
You can use the **xlc_configure** utility to facilitate the use of XL C/C++ with IBM Advance Toolchain. For details, see "Using IBM XL C/C++ for Linux, V13.1.5 with the Advance Toolchain" in *XL C/C++ Compiler Reference*.

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 you find 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.

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 PDF1 step.

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 cleanpdf, mergepdf, and showpdf in the *XL C/C++ Compiler Reference*.

Advance Toolchain 9.0 support

IBM XL C/C++ for Linux, V13.1.5 fully supports IBM Advance Toolchain 9.0, 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, V13.1.5 with Advance Toolchain" 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® processors
- Improving the usage of the memory subsystem

Related information

Optimizing your applications

Optimization and tuning

Compiler built-in functions

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

The parallel programming facilities are based on the concept of threads. Parallel programming exploits the advantages of multiprocessor systems while maintaining a full binary compatibility with existing uniprocessor systems. This means that a multithreaded program that works on a uniprocessor system can take advantage of a multiprocessor system without recompiling.

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 block of code. The existence of the directives in the source removes the need for the compiler to perform any dependence analysis on the parallel code. OpenMP directives require the presence of Pthread libraries to provide the necessary infrastructure for parallelization.

OpenMP directives address the following important issues of parallelizing an application:

1. Clauses and directives are available for scoping variables. Generally, variables should not be shared; that is, each thread should have its own copy of the variable. You cannot declare the scope of C99 variable length array (VLA) variables in `list` of the `private(list)` clause to be private to each thread.
2. Work sharing directives specify how the work contained in a parallel region of code should be distributed across the threads.
3. Directives are available to control synchronization between threads.

Starting from IBM XL C/C++ for Linux, V13.1.5, XL C/C++ supports OpenMP API Version 4.0 and selected features of the OpenMP API Version 4.5 specification. For details, see “OpenMP support” on page 21.

Related information

Optimizing your applications

The OpenMP API specification for parallel programming

Diagnostic reports

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

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 obtain diagnostic information from the compiler in XML or HTML format. The XML and HTML reports provide information about optimizations that the compiler performed or could not perform. 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.

The debugging information can be examined by **gdb** or any symbolic debugger that supports the DWARF debug format on Linux to help you debug your programs.

Related information

-g

Chapter 2. What's new for IBM XL C/C++ for Linux, V13.1.5

This section describes features and enhancements added to IBM XL C/C++ for Linux, V13.1.5.

Built-in functions

This section describes built-in functions that are new or changed for IBM XL C/C++ for Linux, V13.1.5.

New built-in functions

Note: The following new built-in functions are valid only when `-qarch(-mcpu)` is set to target POWER9 processors.

vec_absd
: Returns a vector that contains the absolute difference of the corresponding elements of the given vectors.

vec_cmpne
: Returns a vector containing the results of comparing each set of the corresponding elements of the given vectors for inequality.

vec_cmpnez
: Returns a vector that contains the results of comparing each set of the corresponding elements of the given vectors for inequality, or the results of testing the corresponding element of given vectors for the value of zero.

vec_cntlz
: Counts the most significant zero bits of each element of the given vector.

vec_cntlz_lsbb
: Counts the leading byte elements of the given vector that have a least significant bit of 0.

vec_cnttz
: Counts the least significant zero bits of each element of the given vector.

vec_cnttz_lsbb
: Counts the trailing byte elements of the given vector that have a least significant bit of 0.

vec_extract_exp
: Returns a vector that contains the exponent of the given vector.

vec_extract_sig
: Returns a vector that contains the significand of the given vector.

vec_first_match_index
: Compares each set of the corresponding elements of the given vectors and returns the first position of equality.

vec_first_match_or_eos_index
: Compares each set of the corresponding elements of the given vectors and returns the first position of equality or the position of the end-of-string terminator *\0*.

vec_first_mismatch_index
Compares each set of the corresponding elements of the given vectors and returns the first position of inequality.

vec_first_mismatch_or_eos_index
Compares each set of the corresponding elements of the given vectors and returns the first position of inequality or the position of the end-of-string terminator `\0`.

vec_insert_exp
Returns a vector that combines the exponents of elements from one vector with the signs and the significands of elements from another vector.

vec_parity_lsbb
Returns a vector that computes parity on the least significant bit of each byte of each element of the given vector.

vec_rlmi
Returns a vector that contains each element of the given vector rotated left and inserted under a mask into another vector.

vec_rlnm
Returns a vector that contains each element of the given vector rotated left and intersected with a mask.

vec_test_data_class
Determines the data class of the elements the given vector.

vec_xl_len
Returns a vector that loads a given number of bytes from the given address.

vec_xst_len
Stores a given byte length of a vector to a given address.

Changed built-in functions

Note: The new argument data type combination of the following changed built-in function is valid only when `-qarch(-mcpu)` is set to target POWER9 processors.

vec_bperm
Gathers up to 16 1-bit values from a quadword or from each doubleword element in the specified order, and places them in the specified order either in the rightmost 16 bits of the leftmost doubleword of the result vector register or in the rightmost 8 bits of each doubleword of the result vector register according to the element types, with the rest of the result set to 0.

Compiler options and pragma directives

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

Compiler options

-mcpu=pwr9 or -mcpu=power9 (-qarch=pwr9)
These suboptions are added to **-mcpu(-qarch)**. Specifying **-mcpu=pwr9** or **-mcpu=power9** (**-qarch=pwr9**) produces object code that contains instructions that run on the POWER9 hardware platforms.

-mtune=pwr9 or -mtune=power9 (-qtune=pwr9)
These suboptions are added to **-mtune(-qtune)**. If you specify **-mtune=pwr9** or **-mtune=power9** (**-qtune=pwr9**), optimizations are tuned for the POWER9 hardware platforms.

GPU **-qofflaod**
The **-qoffload** option is added to enable support for offloading OpenMP target regions on an NVIDIA GPU. GPU

GPU **-qpath=@, -qpath=n, -qpath=s, -qpath=w, -qpath=x**
These **-qpath** suboptions are added to specify substitute path names for the following compiler components:

- The PTX assembler
- The NVIDIA C compiler
- The XL intermediate language (W-Code) splitter
- The XL intermediate language (W-Code) to NVVM-IR translator
- The NVVM-IR to PTX translator GPU

GPU **-X (-W)**
The **-X (-W)** option is updated to pass one or more options to specific compiler components that are used during offloading compilation on an NVIDIA GPU. GPU

Pragma directives

IBM XL C/C++ for Linux, V13.1.5 adds support for more OpenMP directives. For these newly supported OpenMP directives, see the summary in "OpenMP support."

OpenMP support

IBM XL C/C++ for Linux, V13.1.5 partially supports the OpenMP Application Program Interface Version 4.5 specification. The XL C/C++ implementation is based on IBM's interpretation of the OpenMP Application Program Interface 4.5.

New directives

In addition to the existing OpenMP directives, IBM XL C/C++ for Linux, V13.1.5 adds support for the following directives and their clauses:

Table 4. OpenMP device constructs

Directive	Clause
omp target data	• if • device • map
omp target enter data	• if • device • map
omp target exit data	• if • device • map

Table 4. OpenMP device constructs (continued)

Directive	Clause
omp target	• defaultmap • device • firstprivate • if • map • private
omp target update	• device • from • if • to
omp declare target	• to
omp teams	• default • firstprivate • num_teams • private • reduction • shared • thread_limit
omp distribute	• collapse • dist_schedule • firstprivate • lastprivate • private
omp distribute parallel for	Any clauses that are accepted by the **omp distribute** or **omp parallel for** directive except the **linear** and **ordered** clauses.

The following directives as combined constructs are also supported. For more information, see OpenMP combined constructs.

- **omp target parallel**
- **omp target parallel for**
- **omp target teams**
- **omp target teams distribute**
- **omp target teams distribute parallel for**
- **omp teams distribute**
- **omp teams distribute parallel for**

For detailed information about these directives, see topic Pragma directives for parallel processing in the *XL C/C++ Compiler Reference*.

New functions

In addition to the existing OpenMP functions, IBM XL C/C++ for Linux, V13.1.5 supports the following OpenMP execution environment functions:

- `omp_get_default_device`
- `omp_get_initial_device`

- `omp_get_num_devices`
- `omp_get_num_teams`
- `omp_get_team_num`
- `omp_is_initial_device`
- `omp_set_default_device`

IBM XL C/C++ for Linux, V13.1.5 also adds support for the following OpenMP device memory functions:

- `omp_target_alloc`
- `omp_target_associate_ptr`
- `omp_target_disassociate_ptr`
- `omp_target_free`
- `omp_target_is_present`
- `omp_target_memcpy`

For detailed information about these OpenMP functions, see OpenMP runtime functions for parallel processing in the *XL C/C++ Compiler Reference*.

New environment variables

IBM XL C/C++ for Linux, V13.1.5 adds support for the following OpenMP environment variables:

- **OMP_DEFAULT_DEVICE = *n***
- **XLSMPOPTS = TARGET = {MANDATORY | OPTIONAL | DISABLE}**

For more information, see topic Environment variables for parallel processing in the *XL C/C++ Compiler Reference*.

Performance and optimization

This topic describes features and enhancements that assist performance tuning and application optimization.

Offloading computations to the NVIDIA GPUs

GPU

You can offload compute-intensive parts of an application and associated data to the NVIDIA GPUs by using the device constructs that are supported by IBM XL C/C++ for Linux, V13.1.5. You must specify the **-qoffload** and **-qsmp** options to enable support for offloading OpenMP target regions to the NVIDIA GPUs.

For more information, see topic Offloading computations to the NVIDIA GPUs in the *XL C/C++ Optimization and Programming Guide*.

GPU

Chapter 3. Migration of your applications

This section lists important considerations when you migrate your applications that were compiled with other versions of XL C/C++ or with GCC.

Migrating from Linux for big endian distributions to Linux for little endian distributions

You must consider some factors when migrating your applications from POWER8® big endian systems.

- To help migrate programs from big endian systems, you can use the **-qaltivec=be** or **-qaltivec=le** option to toggle the vector element sequence in registers to big endian or little endian element order.
- Starting from IBM XL C/C++ for Linux, V13.1.2, legacy macros C++ `__IBMCPP__`, `__xlC__`, `__xlC_ver__` C++ , C `__IBMC__`, and `__xlc__` C are predefined by the compiler with the **-qxlcompatmacros** option. This option helps you migrate programs from IBM XL C/C++ for Linux, V13.1 or earlier for big endian distributions to the current compiler version for little endian distributions.

  Related information

  Program migration from big-endian systems

  -maltivec (-qaltivec)

  -qxlcompatmacros

Migrating from GCC to XL C/C++

When you migrate programs from GCC to V13.1.2 or later releases of XL C/C++ for Linux for little endian distributions, it is recommended that you use the **-qnoxlcompatmacros** option to undefine legacy macros C++ `__IBMCPP__`, `__xlC__`, `__xlC_ver__` C++ , C `__IBMC__`, and `__xlc__` C .

IBM XL C/C++ for Linux, V13.1.5 for little endian distributions predefines these legacy macros with the **-qxlcompatmacros** option. This option helps you migrate programs from Linux for big endian distributions to Linux for little endian distributions. However, when you migrate programs from GCC to V13.1.2 or later releases of XL C/C++ for Linux for little endian distributions, these legacy macros, if defined, might change your source code and result in compilation failure. Therefore, it is recommended that you use the **-qnoxlcompatmacros** option to undefine these legacy macros.

Related information

-qxlcompatmacros

Migrating applications that use transactional memory built-in functions

Starting from IBM XL C/C++ for Linux, V13.1.2, to use transactional memory built-in functions, you must include a header file in the source code. In addition, if you used numeric return values of the transaction begin and end built-in functions, you must replace numeric return values with macro return values that are provided by IBM XL C/C++ for Linux, V13.1.5.

New header file needed for transactional memory built-in functions

You must include the `htmxlintrin.h` file in the source code if you use any of the transactional memory built-in functions.

Changed return values of the transaction begin and end built-in functions

The return values of the transaction begin and end built-in functions are no longer numeric. You must update your program using the following return values:

__TM_begin
This function returns _HTM_TBEGIN_STARTED if successful; otherwise, it returns a different value.

__TM_end
This function returns _HTM_TBEGIN_STARTED if the thread is in the transactional state before the instruction starts; otherwise, it returns a different value.

__TM_simple_begin
This function returns _HTM_TBEGIN_STARTED if successful; otherwise, it returns a different value.

Related information

Transactional memory built-in functions

Transactional memory built-in functions

Mixing object files compiled with different compilers

Most object files that were compiled with different compilers can be linked together. However, under some circumstances, object files are not compatible and must be recompiled.

Note the following restrictions:

- There is no binary compatibility among AIX, Linux for big endian distributions, and Linux for little endian distributions.
- Do not mix object files that were compiled with the big endian compiler and object files that were complied with the little endian compiler.
- Do not mix object and library files that were compiled with different versions of a compiler if the **-qipa** option was used during the compilation. The **-qipa** option instructs the compiler to perform an IPA link for these object and library files. An IPA link might not be able to handle mismatched versions.

Related information:

-qipa

Using interprocedural analysis

Resolving the compatibility issues of IPA object files

It is recommended that you use the latest version of the compiler to compile and link the IPA object files to avoid compatibility issues. If any compatibility issues occur, you can try these resolutions.

IPA object files that are compiled using earlier versions but are linked by a newer version

When IPA object files that are compiled with earlier versions of compilers are linked by a newer version, errors might occur if the IPA object is compiled by one of the following compilers.
- XL Fortran, V15.1.2 or earlier
- XL C/C++, V13.1.2 or earlier

Try resolving the compatibility issue using one of the following methods:
- Recompile and link your object files with the latest XL compiler if you want to use IPA.
- Do not enable the **-qipa** option.

IPA object files that are compiled using newer versions but are linked by an earlier version

If IPA object files that are compiled with newer versions of compilers are linked by an earlier version, errors occur during the link step. You might be able to resolve the issue by recompiling and linking the IPA object files with the latest XL compiler.

For more information, see Using interprocedural analysisUsing interprocedural analysis .

Chapter 4. Enhancements added in earlier releases

This section describes enhancements added in earlier releases. These enhancements also apply to the current release.

Enhancements added in Version 13.1.4

This section describes features and enhancements added in IBM XL C/C++ for Linux, V13.1.4. These features and enhancements apply to later releases as well.

Built-in functions

This section describes built-in functions that are new for IBM XL C/C++ for Linux, V13.1.4.

New built-in functions

The following GCC vector built-in functions are supported:

- vec_vsx_ld
- vec_vsx_st
- vec_xxsldi
- vec_dstt
- vec_xxpermdi

For the mappings between vector built-in functions in GCC and IBM XL C/C++ for Linux, see Supported GCC vector built-in functions.

Compiler options

This topic describes new or changed compiler options.

-fstack-protector (-qstackprotect)
The **-fstack-protector (-qstackprotect)** option is added to provide protection against malicious input data or programming errors that overwrite or corrupt the stack.

-Ofast The **-Ofast** option is added to imply **-O3 -qhot -D__FAST_MATH__**.

-qpdf1, -qpdf2
When **-qpdf1** or **-qpdf2** is specified without a suboption, **exename** is the default suboption. When **-qpdf1=exename** is in effect, the generated PDF file is named `.<output_name>_pdf` by default, where `<output_name>` is the name of the executable file.

-qslmtags
The **-qslmtags** option is added to control whether IBM Software License Metric (SLM) Tags logging tracks compiler license usage.

Starting from IBM XL C/C++ for Linux, V13.1.4, the following GCC options are also supported. For a complete list of the GCC options that are supported, see Supported GCC options.

- -fsemantic-interposition, -fno-semantic-interposition

Note: The default option is **-fno-semantic-interposition** for IBM XL C/C++ for Linux, while the default option is **-fsemantic-interposition** for GCC.

- -maltivec=be
- -maltivec=le
- -Wstack-protector

Performance and optimization

This topic describes features and enhancements that assist performance tuning and application optimization.

Dumping snapshot PDF profiling information to files during execution

When using profile-directed feedback (PDF) optimization in previous releases, you could get PDF profiling information that is written to one or more PDF files only upon normal termination, and no PDF files could be generated upon abnormal termination. Now you can dump PDF profiling information to one or more PDF snapshot files during execution. This is especially useful if you want to save the generated PDF profiling information when the application is to be terminated abnormally. After defining the new environment variable PDF_SIGNAL_TO_DUMP, you can use it to trigger dumping PDF profiling information.

For more information, see "Dumping snapshot PDF profiling information to files during execution" in the *XL C/C++ Optimization and Programming Guide*.

Other enhancements

This section describes other features and enhancements in IBM XL C/C++ for Linux, V13.1.4.

Support for Clang plug-ins to run extra user-defined actions during compilation

To use Clang plug-ins, you must build your plug-ins by using Clang 3.8. For more information, see Running user-defined actions by using Clang plug-ins.

New fundamental or built-in types

The GCC integer scalar type `__int128` is supported in IBM XL C/C++ for Linux, V13.1.4. Use `__int128` or `signed __int128` as the type specifier for a signed 128-bit integer variable. Use `unsigned __int128` as the type specifier for an unsigned 128-bit integer variable.

Enhancements added in Version 13.1.3

This section describes features and enhancements added in IBM XL C/C++ for Linux, V13.1.3. These features and enhancements apply to later releases as well.

C++11 features

In addition to existing C++11 features, this topic lists the C++11 features that are introduced in this release of XL C/C++.

- Atomics
- `thread_local`

With this release, XL C/C++ is now fully compliant with the C++11 standard.

Related information

- Standard features in the *XL C/C++ Language Reference*

C11 features

This topic lists the C11 features that are introduced in this release of XL C/C++. These features are optional in the C11 standard.

- `_Thread_local`
- Atomics

Note: For atomics, the compiler does not support the following data types:

- Atomic floating point types
- Atomic complex and vector types
- Atomic structure types that are packed through pragmas, attributes, or options

Related information

- Standard features in the *XL C/C++ Language Reference*

OpenMP support

IBM XL C/C++ for Linux, V13.1.3 fully supports the OpenMP Application Program Interface Version 3.1 specification and partially supports the OpenMP Application Program Interface Version 4.0 and 4.5 specifications. The XL C/C++ implementation is based on IBM's interpretation of the OpenMP Application Program Interface Version 3.1, 4.0, and 4.5.

In addition to existing OpenMP functions, this version of XL C/C++ now supports the following OpenMP 4.5 functions:

- `omp_get_num_places`
- `omp_get_partition_num_places`
- `omp_get_partition_place_nums`
- `omp_get_place_num_procs`
- `omp_get_place_proc_ids`
- `omp_get_place_num`

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

- The `omp_get_proc_bind` function
- The **OMP_PLACES** environment variable

The following environment variables are extended to control the thread affinity policy:

- **OMP_DYNAMIC**
- **OMP_DISPLAY_ENV**
- **OMP_PROC_BIND**
- **OMP_THREAD_LIMIT**

Built-in functions

This section describes built-in functions that are new or changed for IBM XL C/C++ for Linux, V13.1.3.

New built-in functions

__builtin___*_chk
Provides a buffer overflow protection mechanism that can prevent some buffer overflow attacks.

__builtin_object_size
Returns a constant number of bytes from the given pointer to the end of the object pointed to if the size of object is known at compile time.

vec_cipher_be
Performs one round of the AES cipher operation on an intermediate state by using a given round key.

vec_cipherlast_be
Performs the final round of the AES cipher operation on an intermediate state by using a given round key.

vec_ncipher_be
Performs one round of the AES inverse cipher operation on an intermediate state by using a given round key.

vec_ncipherlast_be
Performs the final round of the AES inverse cipher operation on an intermediate state by using a given round key.

vec_pmsum_be
Performs an exclusive-OR operation by implementing a polynomial addition on each even-odd pair of the polynomial multiplication result of the corresponding elements.

vec_sbox_be
Performs the SubBytes operation, as defined in *Federal Information Processing Standards FIPS-197*, on a given state.

vec_shasigma_be
Performs a secure hash computation in accordance with *Federal Information Processing Standards FIPS-180-3*.

Changed built-in functions

vec_mule
vec_mule now supports `vector signed int` and `vector unsigned int` types of parameters.

vec_mulo
vec_mulo now supports `vector signed int` and `vector unsigned int` types of parameters.

vec_xl Parameter `b` in `vec_xl(a, b)` can now be a `const` pointer. It can also be a vector pointer.

vec_xl_be
Parameter `b` in `vec_xl_be(a, b)` can now be a `const` pointer. It can also be a vector pointer.

vec_xst
Parameter `c` in `vec_xst(a, b, c)` can now be a vector pointer.

vec_xst_be
Parameter `c` in `vec_xst_be(a, b, c)` can now be a vector pointer.

Compiler options and pragma directives

This section describes new or changed compiler options.

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 and pragma directives, see the *XL C/C++ Compiler Reference*.

Compiler options

-fstandalone-debug
The **-fstandalone-debug** option is added to control whether to generate the debugging information for all symbols when used with the **-g** option.

-qidirfirst
The **-qidirfirst** option is added to control whether the compiler searches for user included files in directories that are specified by the **-I** option before searching any other directories.

-qinline
The **-qinline+<***function_name***>** and **-qinline-<***function_name***>** options are added to control whether the named functions must be inlined or must not be inlined.

-qrestrict
The **-qrestrict** option is added to apply the `restrict` type qualifier to the pointer parameters within all functions without modifying the source file.

Pragma directives

#pragma omp atomic
#pragma omp atomic is extended to support sequentially atomic operations by specifying a new optional clause **seq_cst**. This clause forces atomically performed operations to include an implicit flush operation without a list.

#pragma omp parallel, #pragma omp parallel for, #pragma omp parallel sections
These pragma directives are extended to support a new clause **proc_bind**. This clause specifies a policy for assigning threads to places within the current place partition.

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

-g

-I

Other enhancements

This section describes other features and enhancements in IBM XL C/C++ for Linux, V13.1.3.

Software License Metric (SLM) Tags logging support

IBM XL C/C++ for Linux, V13.1.3 fully supports IBM Software License Metric (SLM) Tags logging so that you can use IBM License Metric Tool (ILMT) to track compiler license usage.

For more information, see "Tracking compiler license usage" in the *XL C/C++ Compiler Reference*.

Enhancements added in Version 13.1.2

This section describes features and enhancements added in IBM XL C/C++ for Linux, V13.1.2. These features and enhancements apply to later releases as well.

C++14 features

C++14 is a new C++ programming language standard. This topic lists the C++14 features that are introduced in this release of XL C/C++.

Note: IBM supports selected features of C++14 standard. 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++14 features is complete, including the support of a new C++14 standard library, the implementation might 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++14 features.

- Polymorphic lambda expressions
- Variable templates

Related information

- Standard features in the *XL C/C++ Language Reference*

C++11 features

In addition to existing C++11 features, this topic lists the C++11 features that are introduced in this release of XL C/C++.

- Alignment support
- `constexpr`
- Explicit overrides and final
- Generalized attributes
- Inheriting constructors
- Local and unnamed types as template arguments
- Monomorphic lambdas expressions
- New character types
- New definitions of POD types
- `noexcept`
- Non-static data member initializers
- Range-based `for`
- Raw string literals
- ref-qualifiers
- Template aliases
- Unicode names (UCN) and unicode literals
- Uniform initialization
- Unrestricted unions
- User-defined literals

Related information

- Standard features in the *XL C/C++ Language Reference*

C11 features

This topic lists the C11 features that are introduced in this release of XL C/C++.

- Complex type initializations
- Composite types for variable length arrays
- Conversions between pointers and floating types
- Generic selection
- Temporary lifetime extensions
- `typedef` redeclarations
- Unicode and UTF-8 literals

Related information

- Standard features in the *XL C/C++ Language Reference*

OpenMP support

IBM XL C/C++ for Linux, V13.1.2 fully supports the OpenMP Application Program Interface Version 3.1 specification and 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 3.1 and 4.0.

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

- Atomic update, atomic capture, and atomic swap
- OMP_DISPLAY_ENV environment variable

Built-in functions

This section describes built-in functions that are new or changed for IBM XL C/C++ for Linux, V13.1.2.

New built-in functions

vec_mergee
Merges the values of even-numbered elements of two vectors.

vec_mergeo
Merges the values of odd-numbered elements of two vectors.

Related information

Compiler built-in functions

Commands

This section describes new, changed, or removed compiler commands.

resetpdf
This command has been removed. It is recommended that you use the **cleanpdf** command instead. The behavior of the **resetpdf** command is the same as that of the **cleanpdf** command. For more information, see -qpdf1, -qpdf2 in the *XL C/C++ Compiler Reference*.

Compiler options

This section describes new or changed compiler options.

-qfloat
The following suboptions are added:

subnormals
: This suboption asserts to the compiler that the code uses subnormal floating point values, also known as denormalized floating point values.

nosubnormals
: This suboption asserts to the compiler that the code does not use subnormal floating point values, also known as denormalized floating point values.

Whether or not you specify this suboption, the behavior of your program will not change, but the compiler uses this information to gain possible performance improvements. To use **-qfloat=subnormals** or **-qfloat=nosubnormals**, you must also specify the **-qarch=pwr8** and **-qtune=pwr8** options.

-qfuncsect
: This option places instructions for each function in a separate section. Placing each function in its own section might reduce the size of your program because the linker can collect garbage per function rather than per object file.

-std (-qlanglvl)
: The following suboptions are added to **-qlanglvl**:

C++14 **extended1y**
: Compilation is based on the C++14 standard, invoking most of the C++11 features and all the currently supported C++14 features.

C11 **stdc11**
: Compilation conforms strictly to the ISO C11 standard.

The following suboptions are added to **-std**:

C++14 **c++1y**
: Compilation is based on the C++14 standard, invoking most of the C++11 features and all the currently supported C++14 features.

C++11 **c++11 | c++0x**
: Compilation conforms strictly to the ISO C++ standard plus amendments, also known as ISO C++11.

C++11 **gnu++11 | gnu++0x**
: Compilation is based on the ISO C++ standard, with some differences to accommodate extended language features.

C++ **gnu++03**
: Compilation is based on the ISO C++98 standard, with some differences to accommodate extended language features.

C11 **c11 | c1x | iso9899:2011**
: Compilation conforms strictly to the ISO C11 standard.

C11 **gnu11**
: Compilation is based on the ISO C11 standard, with some differences to accommodate extended language features.

-qlistfmt
: This option creates an XML or HTML report to assist with finding optimization opportunities.

-qstrict=guards
: The actions that are performed by XL C/C++ if you specify the

-qstrict=guards option have been increased. When the **-qstrict=guards** option is in effect, the compiler behavior is as follows:

- The compiler does not move operations past guards.
- When the compiler encounters `if` statements that contain pointer wraparound checks that can be resolved at compile time, it does not remove the checks or the enclosed operations.

-qxlcompatmacros | -qnoxlcompatmacros
These option define or undefine the macros that are used by earlier versions of the compiler for indicating the XL C/C++ compiler product.

Chapter 5. Setting up and customizing XL C/C++

This section describes how to set up and customize the compiler according to your own requirements.

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 do not affect the settings in your custom configuration files.

Related information

Using custom compiler configuration files

Chapter 6. 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:
- Before you use the compiler, ensure that XL C/C++ is properly installed and configured. For more information, see the *XL C/C++ Installation Guide*.
- To learn about writing C/C++ programs, refer to the *XL C/C++ Language Reference*and the C and C++ language standards.

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 might 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. Assembling 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 **-ftime-report(-qphsinfo)**.

Editing C/C++ source files

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

Source programs must be saved using a recognized file name suffix. See “XL C/C++ input and output files” on page 33 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*and the C and C++ language standards.

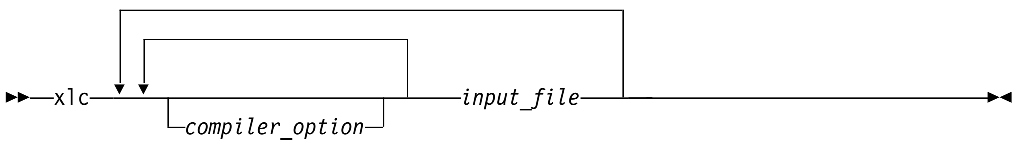
Compiling with XL C/C++

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.

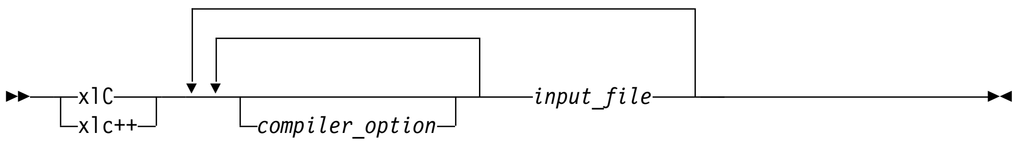
Invoking the compiler

The compiler invocation commands perform all necessary steps to compile C/C++ source files or preprocessed files (`.i` or `.ii`), 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** or **xlC**. 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.

All the compiler invocation commands produce threadsafe code.

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

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
- In your source code using directive statements
- In a makefile
- In the stanzas found in a compiler configuration file

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

Priority sequence of compiler options

Option conflicts and incompatibilities might occur when multiple compiler options are specified. To resolve these conflicts in a consistent manner, the compiler applies the following general priority sequence to most options:

1. Directive statements in your source file override command line settings.
2. Compiler option settings on the command line override configuration file settings.
3. Configuration file settings override default settings.

Generally, if the same compiler option is specified more than once on the command line when the compiler is invoked, 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).

Related information

Compiler options reference

XL C/C++ input and output files

The topic describes the file types that 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 5. Input file types

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

Table 6. 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 `xlc++` to compile `file1.C` and `file3.C` to produce object files `file1.o` and `file3.o`; after that, 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
```

Related information

Linking

Constructing a library

Dynamic and static linking

You can use XL C/C++ 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 might 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. This form of linking is the default and no additional options are needed.

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.

Note: Dynamically and statically linked programs 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 with runtime arguments on the command line.

Canceling execution

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

To cancel a running program, press **Ctrl+C** 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 about 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 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 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 symbolic debugger that supports the DWARF debug format on Linux to step through and inspect the behavior of your compiled application.

Optimized applications pose special challenges when you debug your applications. For more information about debugging your optimized code, see "Debugging optimized code" 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 **--version** (**-qversion**) compiler option.

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

```
xlc++ --version
```

Appendix. Accessibility features for IBM XL C/C++ for Linux

Accessibility features assist users who have a disability, such as restricted mobility or limited vision, to use information technology content successfully.

Accessibility features

IBM XL C/C++ for Linux uses the latest W3C Standard, WAI-ARIA 1.0 (http://www.w3.org/TR/wai-aria/), to ensure compliance to US Section 508 (http://www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-section-508-standards/section-508-standards) and Web Content Accessibility Guidelines (WCAG) 2.0 (http://www.w3.org/TR/WCAG20/). To take advantage of accessibility features, use the latest release of your screen reader in combination with the latest web browser that is supported by this product.

The IBM XL C/C++ for Linux online product documentation in IBM Knowledge Center is enabled for accessibility. The accessibility features of IBM Knowledge Center are described at http://www.ibm.com/support/knowledgecenter/doc/kc_help.html#accessibility.

Keyboard navigation

This product uses standard navigation keys.

Interface information

You can use speech recognition software like a Text-to-speech (TTS) tool to view the output generated by the compiler.

The IBM XL C/C++ for Linux online product documentation is available in IBM Knowledge Center, which is viewable from a standard web browser.

PDF files have limited accessibility support. With PDF documentation, you can use optional font enlargement, high-contrast display settings, and can navigate by keyboard alone.

To enable your screen reader to accurately read syntax diagrams, source code examples, and text that contains the period or comma PICTURE symbols, you must set the screen reader to speak all punctuation.

Related accessibility information

To learn the accessibility features of the operation systems that are supported by IBM XL C/C++ for Linux, see the following information:

- Ubuntu (https://help.ubuntu.com/community/Accessibility)
- SUSE Linux Enterprise Server (https://www.suse.com)
- Red Hat Enterprise Linux (http://www.redhat.com)
- Community Enterprise Operating System (https://www.centos.org/)

In addition to standard IBM help desk and support websites, IBM has established a TTY telephone service for use by deaf or hard of hearing customers to access sales and support services:

TTY service
800-IBM-3383 (800-426-3383)
(within North America)

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see IBM Accessibility (www.ibm.com/able).

Notices

Programming interfaces: Intended programming interfaces allow the customer to write programs to obtain the services of IBM XL C/C++ for Linux.

Trademarks

Index

Product Number: 5765-J08; 5725-C73

Printed in USA

GI13-2875-04