IBM Open XL C/C++ for AIX 17.1.1

Migration Guide

GC28-3309-01

Note

Before using this information and the product it supports, read the information in "Notices" on page 73.

First edition

This edition applies to IBM® Open XL C/C++ for AIX® 17.1.1 (Program 5765-J18; 5725-C72) 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 migration considerations applicable to IBM Open XL C/C++ for AIX 17.1.1.

Who should read this document

This document is intended for C and C++ developers who are to use IBM Open XL C/C++ for AIX 17.1.1 to compile programs that were previously compiled on different platforms, by previous IBM XL C/C++ releases, or by other compilers.

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 v.

While this document covers migration considerations applicable to IBM Open XL C/C++ for AIX 17.1.1, it does not include the following topics:

- An executive overview of new functions: see the *What's New for IBM Open XL C/C++*.
- Compiler installation: see the *IBM Open XL C/C++ Installation Guide*.
- Compiler features including options, pragmas, and built-in functions: see the *IBM Open XL C/C++ User's Guide* for detailed information about the usage of compiler features.

Conventions

Typographical conventions

The following table shows the typographical conventions used in the IBM Open XL C/C++ for AIX 17.1.1 documentation.

Table 1. Typographical conventions

Typeface	Indicates	Example
bold	Lowercase commands, executable names, compiler options, and directives.	The compiler provides basic invocation commands, **ibm-clang**, **ibm-clang_r**, and **ibm-clang++_r**, 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: `ibm-clang myprogram.c -O3`.

Qualifying elements (icons)

Most features described in this documentation 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 documentation uses icons to delineate segments of text as follows:

Table 2. Qualifying elements

Icon	Short description	Meaning
C ▶ ◀ C	C only begins / 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++ ▶ ◀ C++	C++ only begins / 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 ▶ ◀ IBM	IBM extension begins / IBM extension ends	The text describes a feature that is an IBM extension to the standard language specifications.

Syntax diagrams

Throughout this information, diagrams illustrate IBM Open 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.

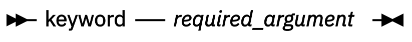
  The ——► symbol indicates that the command, directive, or statement syntax is continued on the next line.

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

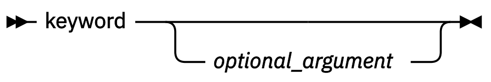
  The ——►◄ symbol indicates the end of a command, directive, or statement.

  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):

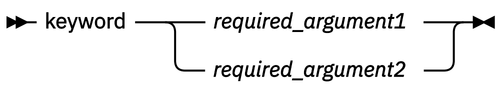


- Optional items are shown below the main path:

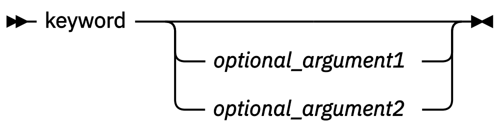


- If you can choose from two or more items, they are shown vertically, in a stack.

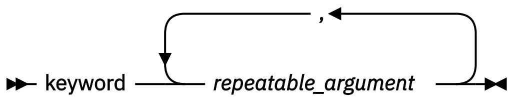
  If you *must* choose one of the items, one item of the stack is shown on the main path.



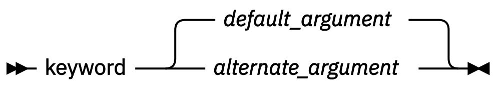
  If choosing one of the items is optional, the entire stack is shown below the main path.



- 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:

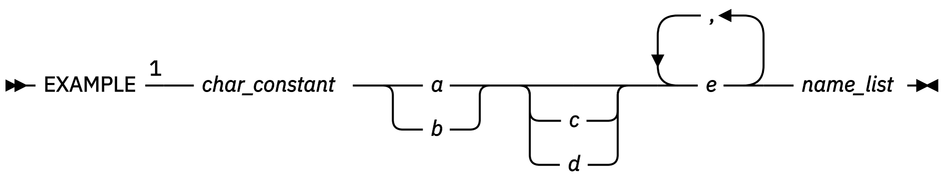


- The item that is the default is shown above the main path.



- 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.

The following is an example of a syntax diagram with an interpretation:



Notes:

[1] IBM extension

Interpret the diagram as follows:

- Enter the keyword EXAMPLE.
- EXAMPLE is an IBM extension.
- 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.
- 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. (The *_list* syntax is equivalent to the previous syntax for *e*.)

How to read syntax statements

Syntax statements are read from left to right:

- Individual required arguments are shown with no special notation.
- When you must make a choice between a set of alternatives, they are enclosed by { and } symbols.
- Optional arguments are enclosed by [and] symbols.
- When you can select from a group of choices, they are separated by | characters.
- Arguments that you can repeat are followed by ellipses (...).

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 documentation

The examples in this documentation, 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.

Related information

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

Available help information

IBM Open XL C/C++ for AIX information

IBM Open XL C/C++ for AIX provides product information in the following formats:

- Quick Start Guide

  The Quick Start Guide (`quickstart.pdf`) is intended to get you started with IBM Open XL C/C++ for AIX 17.1.1. It is located by default in the IBM Open XL C/C++ for AIX directory.
- README files

  README files contain late-breaking information, including changes and corrections to the product information. README files are located by default in the IBM Open XL C/C++ for AIX directory.
- Online product documentation

  The fully searchable HTML-based documentation is viewable in IBM Documentation at https://www.ibm.com/docs/openxl-c-and-cpp-aix/17.1.1.
- PDF documents

  PDF documents are available online at https://www.ibm.com/docs/openxl-c-and-cpp-aix/17.1.1?topic=pdf-format-documentation.

  The following files comprise the full set of IBM Open XL C/C++ for AIX product information.

  Note: To ensure that you can access cross-reference links to other IBM Open XL C/C++ for AIX PDF documents, download and unzip the .zip file that contains all the product documentation files, or you can download each document into the same directory on your local machine.

Table 3. IBM Open XL C/C++ for AIX PDF files

Document title	PDF file name	Description
What's New for IBM Open XL C/C++ for AIX 17.1.1, SC28-3310-01	`whats_new.pdf`	Provides an executive overview of new functions in the IBM Open XL C/C++ for AIX 17.1.1 compiler, with new functions categorized according to user benefits.

Table 3. IBM Open XL C/C++ for AIX PDF files (continued)

Document title	PDF file name	Description
IBM Open XL C/C++ for AIX 17.1.1 Installation Guide, GC28-3311-01	`install.pdf`	Contains information for installing, upgrading, and uninstalling IBM Open XL C/C++ for AIX.
IBM Open XL C/C++ for AIX 17.1.1 Migration Guide, GC28-3309-01	`migrate.pdf`	Contains migration considerations for using IBM Open XL C/C++ for AIX to compile programs that were previously compiled on different platforms, by previous IBM Open XL C/C++ for AIX releases, or by other compilers.
IBM Open XL C/C++ for AIX 17.1.1 User's Guide, SC28-3312-01	`user.pdf`	Contains information about basic compiler usage, various compiler options, pragmas, macros, built-in functions, and 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.

For more information about the compiler, see C/C++ and Fortran compilers on the IBM Power® community at http://ibm.biz/openxl-power-compilers.

Other IBM information

- *Parallel Environment for AIX: Operation and Use*
- The IBM Systems documentation, at https://www.ibm.com/docs/aix, is a resource for AIX information.

  You can find the following books for your specific AIX system:

  - *AIX Commands Reference, Volumes 1 - 6*
  - *Technical Reference: Base Operating System and Extensions, Volumes 1 & 2*
  - *AIX National Language Support Guide and Reference*
  - *AIX General Programming Concepts: Writing and Debugging Programs*
  - *AIX Assembler Language Reference*

Open-source community documentation

- libc++ documentation
- Clang documentation
- LLVM documentation
- LLVM Release Notes
  - Libc++ 15.0.1 - 15.0.7 Release Notes
  - Libc++ 15.0.0 Release Notes
  - Libc++ 14.0.0 Release Notes
  - Libc++ 13.0.0 Release Notes
  - Libc++ 12.0.0 Release Notes
  - Libc++ 11.0.0 Release Notes
  - Libc++ 10.0.0 Release Notes
  - Libc++ 9.0.0 Release Notes
  - Libc++ 8.0.0 Release Notes

Other information

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

Standards and specifications

IBM Open 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 9899:2017*, also known as *C17*.
- *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*.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2017*, also known as *C++17*.
- *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 https://www.nxp.com/docs/reference-manual/ALTIVECPIM.pdf.
- *ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985*.

Technical support

Additional technical support is available from the IBM Open XL C/C++ Support page at https://www.ibm.com/mysupport/s/topic/0TO0z0000006v6TGAQ/xl-cc?productId=01t0z000007g72LAAQ. This page provides a portal with search capabilities to a large selection of Technotes and other support information.

If you have any question on the product, raise it in the IBM C/C++ and Fortran compilers on Power community or open a case at https://www.ibm.com/mysupport/s/topic/0TO0z0000006v6TGAQ/xl-cc?productId=01t0z000007g72LAAQ.

For the latest information about IBM Open XL C/C++ and IBM XL C/C++, visit the product information site at https://www.ibm.com/products/open-xl-cpp-aix-compiler-power.

How to send your comments

Your feedback is important for helping IBM to provide accurate and high-quality information. If you have any comments or questions about this document or any other IBM Open XL C/C++ documentation, send an email to compinfo@cn.ibm.com.

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

Inclusive language

As other industry leaders join IBM in embracing the use of inclusive language, IBM will continue to update the documentation, product code, and user interfaces to reflect those changes. While IBM values the use of inclusive language, terms that are outside of IBM's direct influence are sometimes required for the sake of maintaining user understanding.

To learn more about this initiative, read the Words matter blog on ibm.com®

Chapter 1. Migrating from Classic XL compilers

When you migrate programs from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1, consider factors such as changed compiler options, built-in functions, and environment variables.

Language support

This topic discusses the support for language standard features and IBM extension features in IBM Open XL C/C++ for AIX 17.1.1.

Language standard features

IBM Open XL C/C++ for AIX 17.1.1 incorporates the LLVM and Clang compiler infrastructure. IBM Open XL C/C++ for AIX 17.1.1 supports all of the **`-std`** options that are available in the LLVM Clang compiler on which it is based, all C language specifications up to and including C17, and all C++ language specifications up to and including C++17. Additionally, IBM Open XL C/C++ for AIX 17.1.1 offers experimental availability of C++20 features. Refer to Enhanced LLVM and Clang support for more information of LLVM and Clang.

Note: C++ library features that are newer than C++14 might not be available in IBM Open XL C/C++ for AIX 17.1.1.

IBM extension features

IBM Open XL C/C++ for AIX 17.1.1 does not support all IBM extension features that are supported by IBM XL C/C++ for AIX 16.1.0 or earlier releases. For details of the IBM extension features, refer to the Language Reference of IBM XL C/C++ for AIX 16.1.0 or earlier releases.

The `constructor` and `destructor` function attributes
When you invoked IBM XL C/C++ for AIX 16.1.0 using the **`xlclang`** or **`xlclang++`** invocation command, the `constructor` and `destructor` function attributes applied to the following types of programs:

- C++ programs
- C programs if they are linked using a C++ link invocation command

IBM Open XL C/C++ for AIX 17.1.1 supports `constructor` and `destructor` function attributes in both C and C++ programs.

Binary compatibility

In general, C++ objects built with IBM Open XL C/C++ for AIX 17.1.1 are binary compatible with C++ objects built with IBM XL C/C++ for AIX 16.1.0 invoked by **`xlclang++`**.

The exception is that object files created by IBM XL C/C++ for AIX 16.1.0 with the **`-qpdf1`** option, when the compiler is invoked by **`xlclang`** or **`xlclang++`**, need to be recompiled. This is because the object files make calls to a different PDF library from the ones used in IBM Open XL C/C++. Refer to Profile Guided Optimization (PGO) for more information.

The implementation of the C++11 language standard requires an update to the `std` library `libc++`, and causes a breakage in C++ binary compatibility. Therefore, C++ object files built with IBM Open XL C/C++ for AIX 17.1.1 are not directly interoperable with C++ object files generated by IBM XL C/C++ for AIX 16.1.0 that is invoked by **`xlC`** or earlier releases. You must recompile your classic programs with IBM Open XL C/C++ for AIX 17.1.1 to solve such binary incompatibility and link object files successfully.

Note: However, the incompatibility does not prohibit coexistence of C++ object files that do not pass objects or exceptions. Refer to the "Compatibility with Classic XL C++ object files" on page 2 section for details.

C object files built with IBM Open XL C/C++ for AIX 17.1.1 are binary compatible with C object files built with IBM XL C/C++ for AIX 16.1.0 and earlier releases when both of the following conditions are satisfied:

- Symbol names with external linkage contain only the dollar sign and characters from the basic character set.
- Bitfields in C programs are declared only with the types allowed by the C standard regardless of any types allowed as implementation-defined.

Compatibility with Classic XL C++ object files

Refer to the following types of object files as using `libc++` ABI, which denotes the C++ standard library implementation on which the object files are based:

- C++ object files built with IBM Open XL C/C++ for AIX. These object files utilize interfaces provided by C++ runtime libraries `libc++.a`, `libc++abi.a`, and `libunwind.a`, which are from the `libc++.rte`, `libc++abi.rte`, and `libunwind.rte` fileset, respectively.
- C++ object files built with IBM XL C/C++ for AIX 16.1.0 that is invoked by **`xlclang++`**. These object files utilize interfaces provided by C++ runtime `libc++.a`, which is from the `libc++.rte` fileset.

Similarly, refer to the following type of object files as using `libC` ABI:

- C++ object files built with IBM XL C/C++ for AIX 16.1.0 that is invoked by **`xlC`** or earlier releases. The object files utilize interfaces provided by C++ runtime `libC.a`, which is from the `xlC.rte` fileset.

Different mangled names
The names of C++ functions and other entities are mangled using different name mangling schemes in libc++ ABI and libC ABI to prevent linking of incompatible code. libc++ ABI utilizes the CXA mangling scheme that is defined in the Itanium C++ ABI specification, while libC ABI uses the IBM proprietary mangling scheme. Consequently, the corresponding symbols for a C++ function in these ABIs are different; for example, the symbol of C++ function `func()` is `_Z4funcv` in libc++ ABI, whereas it is `func__Fv` in libC ABI. In the CXA mangling scheme, mangled names have the _Z prefix.

Different object model and layout
The C++ object model and layout differ between libc++ ABI and libC ABI. Accessing C++ objects from one C++ ABI in code that is compiled in another C++ ABI results in undefined behavior, except where the type is subject to rules for compatibility with C.

Different exception handling
The implementation of exception handling differs between libc++ ABI and libC ABI. There are exception handling limitations when exceptions are thrown from the libc++ ABI side and caught or unwound through the libC ABI side, and vice versa. Refer to "Exception compatibility" on page 59 for details.

Due to these differences, libc++ ABI and libC ABI are incompatible and can be considered as different languages, even though both are referred to as C++. However, they can still coexist in the same application with limitations. In such cases, there can be instances of different C++ runtimes coexisting in the same process space without interacting with each other. For example, input and output are buffered separately and are not visible to the other runtime.

Libraries built with coexisting shared objects

Although libc++ ABI and libC ABI are incompatible, you can package their shared objects into a single library so that the library can be used for an application built with either libc++ ABI or libC ABI if all of the following conditions are met:

- The libc++ ABI or libC ABI shared objects in the library are mutually exclusive.
- Each shared object in the library that presents a C++ API supports only one of libc++ ABI and libC ABI.
- Exported symbols from these ABIs are disjoint.
- There are no cross references between libc++ ABI and libC ABI.

If the libc++ ABI and libC ABI shared objects have cross references, or if there is an executable file referencing symbols from two types of shared objects, the shared objects will be linked with the resulting executable file. This results in two different C++ runtime implementations coexisting in the same process space, which might cause unexpected behavior. To confirm the load dependencies of an executable, use the ldd command on the executable.

Mangled C++ symbols in libc++ ABI and libC ABI are disjoint. To prevent symbol name overlapping across ABIs of C/C++ shared objects, it is recommended that symbols such as global variables and C functions be renamed to encode the ABI in the symbol names. For example, you can map a function name to different names either by changing the identifier or by using an `asm` label. In the following example, name `externCFunctionV2` is used for the `externCFunction` function in libc++ ABI and its original name `externCFunction` is used in libC ABI.

```
#if defined(_AIX) && defined(__clang__)
  extern "C" void externCFunction(void) asm("externCFunctionV2");
#else
  extern "C" void externCFunction(void);
#endif
```

Export only the desired symbols from these shared objects and ensure that exported symbols across C++ ABIs in your shared objects do not overlap. To learn about how to export symbols, refer to `Symbol exports and visibilities`. When a C++ ABI is no longer needed for linking new applications but you want to retain load compatibility, a shared object can be made load-only with the **`strip -e`** command.

Unlike shared objects, where exported symbols are controlled by export lists or attribute visibility, having static archive members for both libc++ ABI and libC ABI in the same archive library might result in unexpected behavior. For example, static constructors for both libc++ ABI and libC ABI are executed by the resulting executable, which might not be expected. It is recommended to not include static archive members for both libc++ ABI and libC ABI in the same archive library unless the behavior is well understood.

Example

The following example shows how to build a dual-ABI library from a single source to support both libc++ ABI and libC ABI:

```
$ cat build_example.sh
#!/usr/bin/ksh
rm -f xlc.shr.o ibmclang.shr.o libfunc.a xlc.a.out ibmclang.a.out
xlC -qmkshrobj func.cpp -o xlc.shr.o -bE:xlc.exp
ibm-clang++_r -shared func.cpp -o ibmclang.shr.o -bE:ibmclang.exp
ar -v -q libfunc.a xlc.shr.o ibmclang.shr.o
xlC main.cpp -o xlc.a.out libfunc.a -blibpath:.:/usr/lib
ibm-clang++_r main.cpp -o ibmclang.a.out libfunc.a -blibpath:.:/usr/lib

$ cat func.hpp
#include <iostream>

#if defined(_AIX) && defined(__clang__)
extern "C" void bar(void) asm("bar2");
#else
extern "C" void bar(void);
#endif

void func(void);

$ cat func.cpp
#include "func.hpp"

#if defined(_AIX) && defined(__clang__)
#define BUILD_COMPILER "build compiler is ibm-clang++_r"
#else
#define BUILD_COMPILER "build compiler is xlC"
#endif

void func(void)
{
  std::cout << "func(): " << BUILD_COMPILER << std::endl;
}

extern "C" void bar(void)
{
  std::cout << "bar(): " << BUILD_COMPILER << std::endl;
}
$ cat main.cpp
```

```
#include "func.hpp"

int main() {
  func();
  bar();
  return 0;
}
$ cat xlc.exp
#!
func__Fv
bar
$ cat ibmclang.exp
#!
_Z4funcv
bar2
$ build_example.sh
$ xlc.a.out
func(): build compiler is xlC
bar(): build compiler is xlC
$ ibmclang.a.out
func(): build compiler is ibm-clang++_r
bar(): build compiler is ibm-clang++_r
$
```

In this example, the `main` program built with either the **`xlC`** or **`ibm-clang++_r`** command can call both the `func` and `bar` functions from the dual-ABI library. Shared object `ibmclang.shr.o` in `libfunc.a` is built using the IBM Open XL C/C++ compiler. The shared object can be used by both IBM Open XL C/C++ for AIX and IBM XL C/C++ for AIX 16.1.0 invoked by **`xlclang++`**. Because `ibmclang.shr.o` has dependencies on the C++ runtime of the IBM Open XL C/C++ compiler, applications generated by IBM XL C/C++ for AIX 16.1.0 invoked by **`xlclang++`** require the appropriate version of the IBM Open XL C/C++ for AIX runtime to be available on the system. Similarly, `ibmclang.shr.o` can also be built using IBM XL C/C++ for AIX 16.1.0 invoked by **`xlclang++`** and used by both IBM Open XL C/C++ for AIX and IBM XL C/C++ for AIX 16.1.0 invoked by **`xlclang++`**.

You can confirm the load dependencies of your executable using the AIX command **`ldd`**. The following is the output from using **`ldd`** on `xlc.a.out` and `ibmclang.a.out` in the example above:

```
$ ldd xlc.a.out
xlc.a.out needs:
     ...
     /usr/lib/libC.a(shr.o)
     ...
     /usr/lib/libC.a(shrcore.o)
     /usr/lib/libC.a(ansi_32.o)
     /usr/lib/libC.a(ansicore_32.o)
$ ldd ibmclang.a.out
ibmclang.a.out needs:
     ...
     /usr/lib/libc++.a(shr2.o)
     /usr/lib/libc++abi.a(libc++abi.so.1)
     /usr/lib/libunwind.a(libunwind.so.1)
     /usr/lib/libc++.a(libc++.so.1)
     ...
$
```

According to the output, `xlc.a.out` has dependencies on the C++ runtime of the classic XL C/C++ compiler, which is `libC.a`, but it does not have dependencies on the C++ runtime of the Open XL C/C++ compiler, which are `libc++.a`, `libc++abi.a`, and `libunwind.a`. On the other hand, `ibmclang.a.out` has dependencies on `libc++.a`, `libc++abi.a`, and `libunwind.a` but does not have dependencies on `libC.a`.

Invocation commands

The classic invocation commands supported by IBM XL C/C++ for AIX 16.1.0 or earlier releases including **`xlc`** and **`xlC`** are not supported in IBM Open XL C/C++ for AIX 17.1.1. The **`xlclang`** and **`xlclang++`** invocation commands are not supported either.

In IBM Open XL C/C++ for AIX 17.1.1, use the **ibm-clang**, **ibm-clang_r**, and **ibm-clang++_r** invocation commands instead.

Note:

- **-pthread** is not implied when the compiler is invoked by **ibm-clang**.
- **ibm-clang_r** replaces **xlc** which is an invocation command in IBM XL C/C++ for AIX 16.1.0 or earlier releases. Similarly, **ibm-clang++_r** replaces **xlC**.

For more information on the classic invocation commands, see classic invocation commands.

Compiler utilities and commands

Consider a number of changes to compiler utilities and commands when you migrate your program from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1.

Supported utilities and commands

This topic discusses utilities and commands that are supported by IBM Open XL C/C++ for AIX 17.1.1.

IBM Open XL C/C++ for AIX 17.1.1 introduces support for new utilities, as well as continuing to support the utilities and commands that were supported by IBM XL C/C++ for AIX 16.1.0, except those described in Unsupported utilities and commands. Find details of all supported utilities and commands in Utilities and commands.

Unsupported utilities and commands

IBM Open XL C/C++ for AIX 17.1.1 no longer supports compiler utilities and commands that are described in this section.

- Utilization reporting tool
- The c++filt name demangling utility
- The linkxlC utility
- The makeC++SharedLib utility
- The genhtml command
- The cleanpdf command
- The mergepdf command
- The showpdf command

Migration considerations of individual compiler utilities

This section lists individual compiler utilities that need to be considered for migration.

linkxlC

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **linkxlC** utility links C++ .o and .a files. It can be used on systems where the compiler is not installed.

In IBM Open XL C/C++ for AIX 17.1.1, **linkxlC** is not supported. Use **ibm-clang++_r** to link Open XL C/C++ binaries instead. Contact your customer service if you need to link Open XL C++ binaries where the **ibm-clang++_r** utility is not available.

To link binaries that are compiled with IBM XL C/C++ for AIX 16.1.0 using **xlC** and binaries that are compiled with IBM Open XL C/C++ for AIX 17.1.1 using **ibm-clang++_r**, you can perform one of the following tasks:

- Link mixed binaries with the **xlC** or V16 **linkxlC** command and add necessary V17 linker options. Run the **ibm-clang++_r -v** command to obtain the linker options that IBM Open XL C/C++ for AIX 17.1.1 provides.
- Link mixed binaries with the **ibm-clang++_r** command and add necessary V16 linker options. Run the **xlC -v** command to obtain the linker options that IBM XL C/C++ for AIX 16.1.0 provides.

makeC++SharedLib

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **makeC++SharedLib** utility permits the creation of C++ shared libraries on systems where the compiler is not installed.

In IBM Open XL C/C++ for AIX 17.1.1, the **makeC++SharedLib** utility is not supported. Use **ibm-clang++_r -shared** to create Open XL C++ shared libraries instead. Contact your customer service if you need to create Open XL C++ shared libraries where the **ibm-clang++_r** utility is not available.

In Open XL, similar to what occurs when **makeC++SharedLib** was used in classic XL compilers, an automatic export list is generated when **-shared** is specified without either an export list or the **-bexp*** export options.

Option mapping

Options of **makeC++SharedLib** are recognized by Open XL as linker options, except for the following ones.

Table 4. Mapping options of MakeC++SharedLib to Open XL linker options

Option of MakeC++SharedLib	Linker option
-e	None. The similar function can be obtained by separately running the **CreateExportList** utility over the input files.
-E<export_list>	-bE:<export_list>
-I<import_list>	-bI:<import_list>
-n	-e
-p	-bcdtors::[priority]
-w	None
-X 32	-m32
-X 64	-m64

Notes:

- In Classic XL compilers, the **-p** option is mandatory in **MakeC++SharedLib** to specify the priority level for the initialization order of static C++ objects declared in the shared object. However, **-bcdtors::[priority]** can be omitted in Open XL. If no priority is specified, Open XL uses the default priority of zero.
- In Open XL compilers, if neither **-m32** nor **-m64** is set, the mode specified by the OBJECT_MODE environment variable takes effect. This behavior is the same as that of classic XL compilers when neither **-X 32** nor **-X 64** is set in **MakeC++SharedLib**.

An option that is meant to be passed to the linker **ld** needs to be preceded by the **-Wl,** option; for example, **-Wl,-eMy_entry**. Note that the comma is included as part of the **-Wl,** option.

Creating shared libraries

To create shared binaries from object files that are compiled with IBM XL C/C++ for AIX 16.1.0 using **xlC** and object files that are compiled with IBM Open XL C/C++ for AIX 17.1.1 using **ibm-clang++_r**, you can perform one of the following tasks:

- Link mixed binaries with the **xlC** or V16 **makeC++SharedLib** command and add necessary V17 linker options. Run the **ibm-clang++_r -v** command to obtain the linker options that IBM Open XL C/C++ for AIX 17.1.1 provides.

- Link mixed binaries with the **`ibm-clang++_r`** command and add necessary V16 linker options. Run the **`xlC -v`** command to obtain the linker options that IBM XL C/C++ for AIX 16.1.0 provides.

Related information

- MakeC++SharedLib
- Utilities and commands
- Linking shared libraries
- ld command

Compiler options

Consider a number of changes to compiler options when you migrate your program from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1.

Supported compiler options

This topic describes Clang, GCC, and IBM compiler options that are supported by IBM Open XL C/C++ for AIX 17.1.1.

Clang and GCC options

The new **ibm-clang**, **ibm-clang_r**, and **ibm-clang++_r** invocation commands accept Clang options. For more information about Clang options, see Clang options.

Meanwhile, GCC options that were supported by IBM XL C/C++ for AIX 16.1.0 are also supported by IBM Open XL C/C++ for AIX 17.1.1. Refer to GCC options for details.

Supported IBM compiler options

IBM Open XL C/C++ for AIX 17.1.1 introduces support for new options.

Additionally, the following options that were supported by IBM XL C/C++ for AIX 16.1.0 are also supported by IBM Open XL C/C++ for AIX 17.1.1.

- -b
- -B
- -bmaxdata
- -brtl
- -c
- -C
- -D
- -e
- -E
- -g
- -I
- -l
- -L
- -o
- -r
- -S
- -U
- -w

- -W

Find details of all supported options in Compiler options.

Unsupported compiler options

The classic XL C/C++ compiler options that usually start with **-q** are not supported in IBM Open XL C/C++ for AIX 17.1.1.

You must remove the specified classic options or replace them with the appropriate Clang options.

Changed compiler options

Some compiler options have been changed in IBM Open XL C/C++ for AIX 17.1.1.

`-maltivec`
In IBM Open XL C/C++ and classic release IBM XL C/C++ for AIX 16.1.0, the `altivec.h` file is not implicitly included when **`-maltivec`** is in effect.

Discrepancies for option defaults

The section shows the discrepancies for compiler option defaults between IBM Open XL C/C++ for AIX 17.1.1 and IBM XL C/C++ for AIX 16.1.0.

Table 5. Option defaults on IBM Open XL C/C++ for AIX 17.1.1 and IBM XL C/C++ for AIX 16.1.0

Default on IBM XL C/C++ for AIX 16.1.0 invoked by xlclang or xlclang++	Default on IBM Open XL C/C++ for AIX 17.1.1
-mcpu=power4	-mcpu=power7
-qnolibansi	-fbuiltin, which is an equivalent option to -qlibansi
-qnortti	-frtti, which is an equivalent option to -qrtti or -qrtti=all
-qnostrict is implied when -O3 is in effect	-ffast-math is not implied when -O3 is in effect
C -std=gnu99	-std=gnu17
C++ -std=gnu++11	-std=gnu++14

Mapping of options

The topic provides a mapping of classic IBM XL C/C++ compiler options and Clang options that have the same or similar functions.

Note: Suboptions of these options do not necessarily have a one-to-one mapping.

Table 6. Mapping of compiler options

Classic options supported by IBM XL C/C++ for AIX 16.1.0	Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1
-#	-###
-+	-x c++
-C!	None; the default compiler behavior is as if -C! was in effect.
-E	-E or -E -x c for files with unrecognized file name suffixes
-f	-Wl,-f

Table 6. Mapping of compiler options (continued)

Classic options supported by IBM XL C/C++ for AIX 16.1.0	**Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1**
-F	None
-G	-shared -Wl,-G Refer to Linking a shared Library with runtime linking for more information.
-ma	None
-O	-O0, -O1, -O2, -O3, or -Ofast
-p	None
-pg	None
-P	-E -P -o *file*.i obtains behavior similar to the classic -P option.
-q32	-m32
-q64	-m64
-qaggrcopy	None; the default compiler behavior is as if -qaggrcopy was in effect.
-qalias=[no]ansi	-f[no-]strict-aliasing
-qalias=*suboption*, where *suboption* is not [no]ansi	None
-qalign=bit_packed	-fpack-struct
-qalignrulefor	None
-qalloca	None
-q[no]altivec	-m[no-]altivec
-qarch	-mcpu
-qasm	-fasm
-qasm_as	None
-qassert	None
-qattr	None
-qbitfields=signed	-fsigned-bitfields
-qbitfields=unsigned	None
-qc_stdinc	-isystem
-qcpp_stdinc	-isystem
-qcache	None
-qchars=signed	-fsigned-char
-qchars=unsigned	-funsigned-char
-qcheck=bounds	-fsanitize=bounds -fsanitize-trap=bounds
-qcheck=divzero	-fsanitize=integer-divide-by-zero -fsanitize-trap=integer-divide-by-zero

Table 6. Mapping of compiler options (continued)

Classic options supported by IBM XL C/C++ for AIX 16.1.0	**Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1**
-qcheck=nullptr	-fsanitize=null -fsanitize-trap=null
-qcinc	None
-qcommon	None
-qcompact	-Os and -Oz
-qconcurrentupdate	None
-qcpluscmt	None; the default compiler behavior is as if -qcpluscmt was in effect.
-qcrt	None; the default compiler behavior is as if -qcrt was in effect.
-qnocrt	-nostartfiles
-qdataimported	-mdataimported
-qdatalocal	-mdatalocal
-qnodatalocal	-mdataimported
-qdbcs	None; the compiler supports UTF-8 source files all the time.
-qdbgfmt=dwarf	-gdwarf-3
-qdbgfmt=dwarf4	-gdwarf-4
-qdbgfmt=stabstring	None
-qdbxextra	-fno-eliminate-unused-debug-types
-qdfp	None
-q[no]digraph	-f[no-]digraphs
-qdirectstorage	None
-qdollar	-fdollars-in-identifiers
-qdpcl	None
"-qdump_class_hierarchy" on page 22	-fdump-class-hierarchy or -Xclang -fdump-record-layouts
-qeh	-fexceptions
-qnoeh	-fignore-exceptions
-qenum	None
-qexpfile	None
-qextchk	None
-qfdpr	None
-qflag	None
-qfloat=[no]dfpemulate	None
-qfloat=[no]fenv	None

Table 6. Mapping of compiler options (continued)

Classic options supported by IBM XL C/C++ for AIX 16.1.0	Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1
-qfloat=[no]fltint	None
-qfloat=[no]fold	None
-qfloat=[no]hscmplx	None
-qfloat=[no]hsflt	None
-qfloat=[no]hssngl	None
-qfloat=maf	-ffp-contract=fast
-qfloat=nomaf	-ffp-contract=off
-qfloat=[no]nans	None
-qfloat=relax	-fno-honor-nans -fno-honor-infinities -fdenormal-fp-math=positive-zero -fno-signed-zeros -fno-rounding-math -freciprocal-math
-qfloat=norelax	-fhonor-nans -fhonor-infinities -fdenormal-fp-math=ieee -fsigned-zeros -frounding-math -fno-reciprocal-math
-qfloat=[no]rndsngl	None
-qfloat=[no]rngchk	None
-qfloat=[no]rrm	-f[no-]rounding-math
-qfloat=[no]rsqrt	None
-qfloat=[no]single	None
-qfloat=[no]spnans	None
-qfloat=[no]subnormals	None
-qflttrap=enable:inexact	None
-qflttrap=enable:invalid	None
-qflttrap=enable:overflow	None
-qflttrap=enable:underflow	None
-qflttrap=enable:zerodivide	-fsanitize=float-divide-by-zero -fsanitize-trap=float-divide-by-zero
-qflttrap=imprecise	None
-qflttrap=nanq	None
-qformat	-Wformat
-qfullpath	None; the default compiler behavior is as if -qfullpath was in effect.
-q[no]funcsect	-f[no-]function-sections
-qfunctrace	None
-qgcc_c_stdinc	None
-qgcc_cpp_stdinc	None

Table 6. Mapping of compiler options (continued)

Classic options supported by IBM XL C/C++ for AIX 16.1.0	**Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1**
-qgenproto	None
-qhalt=w	-Werror
-qhaltonmsg	None
-qheapdebug	None
-qhelp	-help
"-qhot" on page 25	-Ofast
-qhot=*suboption*	None
-qidirfirst	None
-qignerrno	-fno-math-errno
-qignprag	None
-qinclude	-include
-qinfo=[no]unset	-W[no-]uninitialized
-qinfo=*suboption*, where *suboption* is not [no]unset	None
-qinitauto	None
-qinlglue	None; the default compiler behavior is as if -qnoinlglue was in effect.
-q[no]inline	-f[no-]inline
-qinline=auto	-finline-functions
-qinline=noauto	-finline-hint-functions
-qinline=level=1	-mllvm --inline-threshold=45 -mllvm --inlinehint-threshold=325
-qinline=level=2	-mllvm --inline-threshold=90 -mllvm --inlinehint-threshold=325
-qinline=level=3	-mllvm --inline-threshold=135 -mllvm --inlinehint-threshold=325
-qinline=level=4	-mllvm --inline-threshold=180 -mllvm --inlinehint-threshold=325
-qinline=level=5	-mllvm --inline-threshold=225 -mllvm --inlinehint-threshold=325
-qinline=level=6	-mllvm --inline-threshold=270 -mllvm --inlinehint-threshold=395
-qinline=level=7	-mllvm --inline-threshold=315 -mllvm --inlinehint-threshold=465
-qinline=level=8	-mllvm --inline-threshold=360 -mllvm --inlinehint-threshold=535
-qinline=level=9	-mllvm --inline-threshold=405 -mllvm --inlinehint-threshold=605

Table 6. Mapping of compiler options (continued)

Classic options supported by IBM XL C/C++ for AIX 16.1.0	Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1
-qinline=level=10	-mllvm --inline-threshold=450 -mllvm --inlinehint-threshold=650
-qipa	-flto
-qisolated_call	None
-qkeepinlines	None
-qkeepparm	None
-qkeyword	None
C -qlanglvl=stdc89	-std=c89 | c90
C -qlanglvl=extc89	-std=gnu89 | gnu90
C -qlanglvl=stdc99	-std=c99
C -qlanglvl=extc99	-std=gnu99
C -qlanglvl=stdc11	-std=c11
C -qlanglvl=extc1x	-std=gnu11
C -qlanglvl=extended	-std=gnu89
C++ -qlanglvl=strict98	-std=c++98/c++03
C++ -qlanglvl=extended	-std=gnu++98 | gnu++03
C++ -qlanglvl=extended0x	-std=c++0x | gnu++11 | gnu++0x
C++ -qlanglvl=extended1y	-std=gnu++1y | gnu++14
-qlanglvl=[no]gnu_warning	-W[no-]#warnings
-qlargepage	None
-qldbl128	None
-qlongdouble	None
-qnolib	-nodefaultlibs
-qnolibansi	-fno-builtin
-qlibmpi	None; the default compiler behavior is as if -qnolibmpi was in effect.
-qlinedebug	-g1 or -gline-tables-only
-qlist	-S
-qlistfmt	None
-qlistopt	None
-qlonglit	None
-qlonglong	None; the default compiler behavior is as if -qlonglong was in effect.
-qmacpstr	-fpascal-strings

Table 6. Mapping of compiler options (continued)

Classic options supported by IBM XL C/C++ for AIX 16.1.0	**Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1**
-qmakedep	None
-qmaxerr	-ferror-limit
-qmaxmem	None
-qmbcs	None; the compiler supports UTF-8 source files all the time.
-qminimaltoc	None
-qmkshrobj	-shared Refer to Linking shared libraries for more information.
-qmkshrobj=*priority*	-bcdtors::*priority*: Refer to Linking shared libraries for more information.
-qnamemangling	None
-qobjmodel	None
-qoldpassbyvalue	None
-qoptdebug	None
-qoptfile	@file
-qoptimize	--optimize
-qpack_semantic=gnu	-fno-xl-pragma-pack
-qpack_semantic=ibm	-fxl-pragma-pack
-qpagesize	None
-qpath	None
-qpdf1	-fprofile-generate
-qpdf2	-fprofile-use
-qphsinfo	-ftime-report
-qpic=small	-fpic
-qpic=large	-fpic -mcmodel=large -Wl,-bbigtoc
-qppline	None
-qnoppline	-E -P
-qprefetch	None
-qprint	None
-qpriority	None
-qprocimported	None
-qproclocal	None
-qprocunknown	None

Table 6. Mapping of compiler options (continued)

Classic options supported by IBM XL C/C++ for AIX 16.1.0	Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1
-qprofile	None
-qproto	None; the default compiler behavior is as if -qnoproto was in effect.
-qreport	None
-qreserved_reg	None
-qrestrict	-frestrict-args
-qro	None; the default compiler behavior is as if -qro was in effect.
-qroconst	None; the default compiler behavior is as if -qroconst was in effect.
-qroptr	None; the default compiler behavior is as if -qnoroptr was in effect.
-q[no]rtti	-f[no-]rtti
-qsaveopt	None
-qshowinc	None
-qshowmacros	-dM
-qshowpdf	None
-qsimd=noauto	-fno-vectorize -fno-slp-vectorize
-qskipsrc	None
-qslmtags	None
-qsmallstack	None
-qsmp	None
-qsource	None
-qsourcetype	-x
-qsourcetype=default	-x none
-qspeculateabsolutes	None
-qspill	None
-q[no]srcmsg	-f[no-]caret-diagnostics
-qstackprotect	-fstack-protector
-qstaticinline	None
-qstaticlink	None
-qstatsym	None
-qstdinc	-nostdinc
-qnostdinc	-nostdinc++
-qstrict=association	-fno-associative-math

Table 6. Mapping of compiler options (continued)

Classic options supported by IBM XL C/C++ for AIX 16.1.0	**Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1**
-qstrict=noassociation	-fassociative-math
-qstrict=[no]nans	-f[no-]honor-nans
-qstrict=[no]infinities	-f[no-]honor-infinities
-qstrict=[no]zerosigns	-f[no-]signed-zeros
-qstrict=subnormals	-fdenormal-fp-math=ieee
-qstrict=nosubnormals	-fdenormal-fp-math=positive-zero or -fdenormal-fp-math=preserve-sign
-qstrict=operationprecision	-fno-reciprocal-math
-qstrict=nooperationprecision	-freciprocal-math
-qstrict=[no]vectorprecision	None
-qstrict=[no]reductionorder	None
-qstrict=[no]guards	None
-qstrict=[no]library	None
-qstrict=order	-fno-associative-math
-qstrict=noorder	-fassociative-math
-qstrict=ieeefp	-fhonor-nans -fhonor-infinities -fdenormal-fp-math=ieee -fsigned-zeros -frounding-math -fno-reciprocal-math
-qstrict=noieeefp	-fno-honor-nans -fno-honor-infinities -fdenormal-fp-math=positive-zero -fno-signed-zeros -fno-rounding-math -freciprocal-math
-qstrict=exceptions	-fhonor-nans -fhonor-infinities -fdenormal-fp-math=ieee -ffp-exception-behavior=strict
-qstrict=noexceptions	-fno-honor-nans -fno-honor-infinities -fdenormal-fp-math=positive-zero -ffp-exception-behavior=ignore
-qstrict=precision	-fno-associative-math -fdenormal-fp-math=ieee -fno-reciprocal-math
-qstrict=noprecision	-fassociative-math -fdenormal-fp-math=positive-zero -freciprocal-math
-qstrict_induction	None
-qsuppress	None
-qsymtab	None
-qsyntaxonly	-fsyntax-only
-qtabsize	None
-qtbtable	None; the default compiler behavior is as if -qtbtable=full was in effect.
-qtempinc	None

Table 6. Mapping of compiler options (continued)

Classic options supported by IBM XL C/C++ for AIX 16.1.0	**Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1**
-qtemplatedepth	-ftemplate-depth
-qtemplaterecompile	None
-qtemplateregistry	None
-qtempmax	None
-qthreaded	-pthread
-qtimestamps	None
-qtls	-ftls-model
-qtmplinst	None
-qtmplparse	None
-qtocdata	None
-qtocmerge	None
-qtrigraph	-trigraphs
-qtune	None
-qtwolink	-bcdtors:mbr
-qunique	None
-q[no]unroll	-f[no-]unroll-loops
-qunroll=yes	None
-qunwind	None
-qupconv	None; the default compiler behavior is as if -qnoupconv was in effect.
-qutf	None
-qvecnvol	-mabi=vec-extabi
-qnovecnvol	-mabi=default
-qversion	--version
-qvisibility	-fvisibility; see also -fvisibility-inlines-hidden.
-qvrsave	None; the default compiler behavior is as if -qnovrsave was in effect.
-qwarn0x	None
-qwarn64	None
-qweakexp	None
-qweaksymbol	None; the default compiler behavior is as if -qweaksymbol was in effect.
-qxcall	None
-qxlcompatmacros	None
-qxref	None

Table 6. Mapping of compiler options (continued)

Classic options supported by IBM XL C/C++ for AIX 16.1.0	Similar/equivalent Clang options supported by IBM Open XL C/C++ for AIX 17.1.1
-s	-Wl,-s
-t	None
-v	-v
-V	-v
-y	None
-Z	-Wl,-Z

Related information

- The "Clang command line argument reference" section in the Clang documentation

Migration considerations for individual compiler options

This section contains migration considerations for individual compiler options.

Related information

- “Mapping of options” on page 8
- “Discrepancies for option defaults” on page 8

-+ (plus sign) (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, you could use the **-+** option to compile any file as a C++ language file. The **-x c++** option, which is supported by IBM Open XL C/C++ for AIX 17.1.1, has an equivalent function that treats input files as C++ source files regardless of their file suffixes.

Note the following differences between new **-x c++** and classic **-+** options:

- The **-+** option is not sensitive to position on the command line. You can specify the input files and the **-+** option in any order on the command line. However, the **-x c++** option affects only the files that are specified following the option on the command line, but not those that precede the option.
- The **-+** option does not accept files that have the `.a`, `.o`, `.so`, `.S`, or `.s` suffixes. However, the **-x** option accepts all such files. For example, if you specify the `a.o` file after **-x c++** on the command line, the compiler treats `a.o` as a c++ file, which might result in errors.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-b

In IBM Open XL C/C++ for AIX 17.1.1, options starting with **-b** that are not otherwise recognized by the compiler are implicitly forwarded to the linker. Linker options with the **-Wl** prefix can be explicitly forwarded to the linker.

-E

Unlike IBM XL C/C++ for AIX 16.1.0 and earlier releases, IBM Open XL C/C++ for AIX 17.1.1 ignores the **-E** option for input files with unrecognized file name suffixes. You can simulate the behavior of classic XL releases by specifying the C language type through the **-x c** option when you have input files with unrecognized file name suffixes.

In the following example, the classic XL compiler preprocesses `file.sqc` using the C language type and emits the preprocessed output to `stdout`:

```
xlc -E file.sqc
```

In IBM Open XL C/C++ for AIX 17.1.1, to obtain similar behavior, specify the C language type before input files that have unrecognized file name suffixes. See the following example:

```
ibm-clang -E -x c file.sqc    #-x -c indicates the C language type
```

-f

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-f** option names a file that stores a list of object files for the compiler to pass to the linker.

In IBM Open XL C/C++ for AIX 17.1.1, use the **-Wl,-f** option instead. However, if the linker is invoked to produce a shared object but an export file is not provided, the resulting shared object might not export the correct symbols. In addition, the **-Wl,-f** option doesn't work if no objects or source files are provided on the command line.

-O

IBM Open XL C/C++ for AIX 17.1.1 supports the **-O0**, **-O1**, **-O2**, **-O3**, and **-Ofast** optimization levels.

Note the following migration considerations for **-O**:

- The **-O1** optimization level does not exist in IBM XL C/C++ for AIX 16.1.0 or earlier releases. This optimization level is less aggressive than **-O2**.
- IBM XL C/C++ for AIX 16.1.0 or earlier releases has the **-O5** optimization level but the level is not available in IBM Open XL C/C++ for AIX 17.1.1.
- The **-O4** level is currently treated as equivalent to **-O3**. However, the behavior of **-O4** might change in the future, so you are recommended not to use **-O4** in IBM Open XL C/C++ for AIX 17.1.1.
- The **-O** optimization level is equivalent to **-O2** in IBM XL C/C++ for AIX 16.1.0 or earlier releases, but it is equivalent to **-O1** in IBM Open XL C/C++ for AIX 17.1.1.
- If you were using **-O4** or **-O5** in IBM XL C/C++ for AIX 16.1.0 or earlier releases, use **-Ofast -mcpu=native -flto** in IBM Open XL C/C++ for AIX 17.1.1 instead.

Related information

- -mcpu

-p, -pg, -qprofile

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-p**, **-pg**, or **-qprofile** option prepares the object files produced by the compiler for profiling. When you compile your program with one of the options, the compiler produces monitoring code that counts the number of times each routine is called. When you execute an application compiled with **-p** or **-qprofile=p**, the application writes the recorded information to a `mon.out` file. When you execute an application compiled with **-pg** or **-qprofile=pg**, the application writes the recorded information to a `gmon.out` file.

In IBM Open XL C/C++ for AIX 17.1.1, the **-p** and **-pg** options are accepted, but the instrumentation does not work correctly. When you execute an application compiled with **-p**, the application links and runs but the generated `mon.out` file does not contain any call information. When you execute an application compiled with **-pg**, the following link error message is issued:

```
ld: 0711-317 ERROR: Undefined symbol: .mcount
```

Profile guided optimization (PGO) is an alternative for the **-pg** option. The **ibm-llvm-profdata show** utility can be used to display information from the PGO profile file.

Related information

- “Profile Guided Optimization (PGO)” on page 63
- Utilities and commands

-P

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-P** option preprocesses the source files specified in the compiler invocation, without compiling, and creates an output preprocessed file for each input file. The preprocessor does not generate line markers.

IBM Open XL C/C++ for AIX 17.1.1 still supports the **-P** option, but its behavior has been changed to be consistent with that of **-P** in GCC. Specifically, **-P** no longer generates preprocessed output files. Instead, it disables the generation of line markers in the preprocessed output generated by other options such as **-E**.

To obtain similar behavior of **-P** in IBM XL C/C++ for AIX 16.1.0 or earlier releases, use the **-E -P** and **-o** options in IBM Open XL C/C++ for AIX 17.1.1.

Example

This is a classic XL command. After executing the command, `file.c` is preprocessed without line markers generated in the preprocessed output.

```
xlc -P file.c
```

This classic XL command can be replaced by the following command in IBM Open XL C/C++:

```
ibm-clang -E -P -o file.i file.c
```

where,

- The **-E** option instructs the compiler to preprocess `file.c` without compiling.
- The **-P** option instructs the compiler to not insert line markers in the preprocessed output.
- The **-o** `file.i` option instructs the compiler to store the preprocessed output to `file.i`.

Related information

- GCC online documentation

-qaggrcopy

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qaggrcopy** option enables destructive copy operations for structures and unions.

IBM Open XL C/C++ for AIX 17.1.1 does not support a functionally equivalent option to **-qaggrcopy**. By default, the compiler uses a destructive copy for structure or union assignments. If a non-destructive copy for structure or union assignments is desired, you are recommended to use the `memmove` library function.

Related information

- movement subroutine

-qalias

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qalias** option indicates whether a program contains certain categories of aliasing or does not conform to C/C++ standard aliasing rules.

In IBM Open XL C/C++ for AIX 17.1.1, classic compiler option **-qalias=ansi** maps to **-fstrict-aliasing** and **-qalias=noansi** maps to **-fno-strict-aliasing**. You can use **-fno-strict-aliasing** to compile code that does not adhere to the ANSI C/C++ standard aliasing rules. However, it is recommended to compile all files with **`-fstrict-aliasing`** during an LTO build to achieve the best performance.

IBM Open XL C/C++ for AIX 17.1.1 does not support a functionally equivalent option to **`-qalias=restrict`**. Optimizations for pointers that are qualified with the C99 `restrict` keyword or the C++ `__restrict` keyword are always enabled in IBM Open XL C/C++ for AIX 17.1.1 and cannot be disabled.

Related information

- Link Time Optimization (LTO)
- GCC online documentation
- The "Clang command line argument reference" section in the Clang documentation

-qalign

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qalign`** option specifies the alignment of data objects in storage, which avoids performance problems with misaligned data.

If you used the **`-qalign`** option to compile your program, remove this option and add the corresponding **`#pragma align`** and **`#pragma align(reset)`** directives in your program when you migrate the program to IBM Open XL C/C++ for AIX 17.1.1.

Related information

- #pragma align

-qalloca, -ma (C only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qalloca`** or **`-ma`** option provides an inline definition of system function `alloca` when it is called from source code that does not include the `alloca.h` header file.

IBM Open XL C/C++ for AIX 17.1.1 does not support a functionally equivalent option to **`-qalloca`** or **`-ma`**. However, you can achieve a similar function using one of the following ways:

- Specify **`-Dalloca=__builtin_alloca`** on the command line to map **`-qalloca`** to the `__builtin_alloca` function. This built-in function can be called without including the `alloca.h` header file.
- Include the `alloca.h` header file in source files.

Related information

- "#pragma alloca (C only)" on page 35
- __builtin_alloca

-qassert

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qassert`** option provides information about the characteristics of the programs that can help to fine-tune optimizations.

In IBM Open XL C/C++ for AIX 17.1.1, there is no option that is functionally equivalent to **`-qassert`**. However, you can get similar optimization hints through the `__builtin_assume()` or `__builtin_assume_aligned()` function.

-qcompact

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qcompact`** option avoids optimizations that increase code size.

In IBM Open XL C/C++ for AIX 17.1.1, the **-Oz** or **-Os** option provides a similar function to control output code size, but **-Oz** and **-Os** work only at the **-O2** optimization level. For example, if you specify **-O3 -Os**, **-O3** is overridden.

When you port your program from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1, reevaluate the size of the output generated by IBM Open XL C/C++ for AIX 17.1.1. If reduction in code size is required, try using the **-Os** option and note any performance trade-offs. If further reduction in code size is required, try using the **-Oz** option.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qc_stdinc (C only), -qcpp_stdinc (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qc_stdinc** or **-qcpp_stdinc** option changes the standard search location for system header files and XL C header files or XL C/C++ header files.

In IBM Open XL C/C++ for AIX 17.1.1, use the **-isystem** option to achieve the same effect.

Note: The directory for the compiler to search for header files needs to be updated for IBM Open XL C/C++ for AIX 17.1.1.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qcinc (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qcinc** option places an `extern "C" { }` wrapper around the contents of header files that are located in a specified directory.

In IBM Open XL C/C++ for AIX 17.1.1, there is no option that is functionally equivalent to **-qcinc**. You need to manually add the `extern "C" { }` wrapper in your program.

-qcpluscmt (C only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qcpluscmt** option enables the recognition of C++-style comments in C source files.

In IBM Open XL C/C++ for AIX 17.1.1, C++ comments are accepted in C source files. There is no way to disable the recognition of C++-style comments in C source files.

-qdump_class_hierarchy

IBM Open XL C/C++ for AIX 17.1.1 does not support an option that is functionally equivalent to **-qdump_class_hierarchy**. In IBM Open XL C/C++ for AIX 17.1.1, **-Xclang -fdump-record-layouts** can be used to produce a similar report of structure layouts; however, the format of the report is different from the report generated by **-qdump_class_hierarchy** in IBM XL C/C++ for AIX 16.1.0 or earlier releases.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qenum

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qenum** option specifies the amount of storage occupied by enumerations.

In IBM Open XL C/C++ for AIX 17.1.1, use **-fshort-enums** as a functionally equivalent option to **-qenum=small**. Other **-qenum** suboptions are not supported. For C++ programs, you can also use the C++11 scoped enumeration feature to specify the underlying type of enumerations.

Related information

- "#pragma enum" on page 36
- GCC online documentation
- The "Clang command line argument reference" section in the Clang documentation

-qexpfile

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qexpfile** option saves all exported symbols in a designated file when used together with the **-qmkshrobj** or **-G** option.

IBM Open XL C/C++ for AIX 17.1.1 does not support a functionally equivalent option to **-qexpfile**. The **CreateExportList** utility that has a similar function to **-qexpfile** is available in IBM Open XL C/C++ for AIX 17.1.1. This utility creates a file that contains a list of all the exportable symbols found in a given set of object files.

-qflag, -qhaltonmsg, -qinfo, -qsuppress

The mechanism of diagnostic message control of Clang is different from that of the classic XL compilers. Refer to the "Diagnostic flags in Clang" section in the Clang documentation for details.

-qfloat

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qfloat** option selects different strategies for speeding up or improving the accuracy of floating-point calculations. The majority of **-qfloat** suboptions do not have functionally equivalent options in IBM Open XL C/C++ for AIX 17.1.1.

If you specified **-qfloat** when compiling your program with IBM XL C/C++ for AIX 16.1.0 or earlier releases, consider the following guidelines when migrating your program to IBM Open XL C/C++ for AIX 17.1.1:

- The **-ffp-model=strict** option ensures correct compiler behavior but disables almost all floating-point optimizations. Use this option with discretion.
- Aspects of **-qfloat=[no]fenv** that pertain to rounding mode can be controlled via **-f[no-]rounding-math**. Similarly, aspects of **-qfloat=[no]fenv** that pertain to exception behaviors can be controlled via **-ffp-exception-behavior**.
- In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qfloat=[no]fold** option could evaluate constant floating-point expressions at compile time. In IBM Open XL C/C++ for AIX 17.1.1, you can use the **-f[no-]rounding-math** option to control constant folding in some cases, but this option affects more than just constant folding and might have performance and accuracy implications that go beyond what **-qfloat=[no]fold** controlled.
- In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qfloat=[no]hscmplx** option could speed up operations that involve complex division and complex absolute value, and the **-qfloat=[no]hsflt** option could speed up calculations by preventing rounding for single-precision expressions and by replacing floating-point division by multiplication with the reciprocal of the divisor. In IBM Open XL C/C++ for AIX 17.1.1, use the **-f[no-]rounding-math** option to control optimizations that might result in different rounding behaviors and use the **-f[no-]reciprocal-math** option to control replacement of divide operations with a multiplication of the numerator by the reciprocal of the denominator.
- In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qfloat=[no]nans** allowed you to use the **-qflttrap=invalid:enable** option to detect and deal with exception conditions that involve signaling NaN (not-a-number) values. In IBM Open XL C/C++ for AIX 17.1.1, use the **-ffp-exception-behavior** option to control some aspects of the semantics controlled by **-qfloat=[no]nans**, but the impact goes beyond just signaling NaNs. Furthermore, unlike IBM XL

C/C++ for AIX 16.1.0 or earlier releases, there is no way to control exceptions when converting an SNaN from single to double precision in IBM Open XL C/C++ for AIX 17.1.1.

- In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qfloat=[no]rsqrt** speeded up some calculations by replacing division by the result of a square root with multiplication by the reciprocal of the square root. In IBM Open XL C/C++ for AIX 17.1.1, use the **-f[no-]reciprocal-math** option to achieve a similar effect, but the impact goes beyond just division by the square root.
- The **-qfloat=[no]spnans** option was deprecated and replaced with **-qfloat=nans**. Consider the migration guidelines of **-qfloat=[no]nans** if you used **-qfloat=[no]spnans** to compile your program.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qflttrap

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qflttrap** option determines what types of floating-point exceptions to detect at runtime.

In IBM Open XL C/C++ for AIX 17.1.1, you can use **-fsanitize=float-divide-by-zero -fsanitize-trap=float-divide-by-zero** to achieve the same effect of **-qflttrap=enable:zerodivide** to detect and trap floating-point divisions by zeros. Other floating-point exception detections using *software traps* that are enabled by **-qflttrap=enable** are no longer supported.

In IBM Open XL C/C++ for AIX 17.1.1, you are recommended to use *hardware traps* to get SIGFPE. You can use facilities such as `fp_trap` and `fp_enable_all` in the program to enable the detection and generation of the SIGFPE signals. Note that the facilities are not portable between platforms so you need to update and recompile your program when migrating the program from AIX to Linux®, or vice versa. In addition, the imprecise and nanq trappings are no longer supported because hardware traps do not need them.

The **-ftrapping-math** option is available in IBM Open XL C/C++ for AIX 17.1.1 to prevent optimizations that can change the trapping behavior of the program, but it does not control whether a signal is generated when a floating-point exception happens.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qfullpath

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-q[no]fullpath** option records the absolute or relative path names of source and header files in object files compiled with debugging information so that debugging tools can correctly locate the source files. When **-qfullpath** is in effect, the absolute path names of source files are preserved. When **-qnofullpath** is in effect, the relative path names of source files are preserved.

In IBM Open XL C/C++ for AIX 17.1.1, only the DWARF debugging information is supported. The default compiler behavior is as if **-qfullpath** were in effect, which means the compiler embeds the absolute paths for source files in the debug information. However, if the source files are not located in the absolute path location, the debugger might resort to the relative path location.

In addition, the **-fdebug-prefix-map** option is a related option to **-qfullpath**, which remaps file source paths in the debug information.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qfunctrace

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qfunctrace** option calls the tracing routines to trace the entry and exit points of the specified functions in a compilation unit.

In IBM Open XL C/C++ for AIX 17.1.1, the **-finstrument-functions** option has a similar function, which generates calls to instrument entry and exit points of functions.

Related information

- "#pragma nofunctrace" on page 40
- The "Clang command line argument reference" section in the Clang documentation

-qhot

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qhot** option performs high-order loop analysis and transformations (HOT) during optimization.

If you specified **-qhot** when compiling your program with IBM XL C/C++ for AIX 16.1.0 or earlier releases, consider the following guidelines when migrating your program to IBM Open XL C/C++ for AIX 17.1.1:

- If you used **-qhot** without any suboption in IBM XL C/C++ for AIX 16.1.0 or earlier releases, use **-Ofast** instead in IBM Open XL C/C++ for AIX 17.1.1.
- If you used **-qnohot**, try using **-Ofast** in IBM Open XL C/C++ for AIX 17.1.1. However, if you encounter any floating-point precision issues, try using **-O3** instead. If the issues can be resolved, you can go back to use **-Ofast** and achieve finer control over the floating-point optimizations with one or more of the following options:
  - **-f[no-]honor-infinities**
  - **-f[no-]honor-nans**
  - **-f[no-]math-errno**
  - **-f[no-]finite-math-only**
  - **-f[no-]associative-math**
  - **-f[no-]reciprocal-math**
  - **-f[no-]signed-zeros**
  - **-f[no-]trapping-math**
  - **-ffp-contract**
  - **-f[no-]rounding-math**
- There is not an option that is functionally equivalent to **-qhot=level=0|1|2**. However, you can control the level of optimizations including some loop optimizations with the **-O2**, **-O3**, or **-Ofast** option.
- The effect of **-qhot=vector** can be achieved by specifying **-mllvm -vector-library=MASSV** along with an optimization level that triggers loop vectorization.
- The effect of **-qhot=fastmath** can be achieved by specifying **-O3 -fapprox-func** or **-Ofast** that invokes scalar MASS library and triggers loop vectorization.

Related information

- **-mllvm**
- The "Clang command line argument reference" section in the Clang documentation

-qignerrno

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qignerrno** option allows the compiler to perform optimizations as if system calls would not modify `errno`. The `__IGNERRNO__` macro is defined to 1 when the **-qignerrno** option is in effect.

In IBM Open XL C/C++ for AIX 17.1.1, the **-fno-math-errno** option provides the same function as **-qignerrno**. However, the `__IGNERRNO__` macro is not supported in IBM Open XL C/C++ for AIX 17.1.1, so **-fno-math-errno** does not predefine `__IGNERRNO__`.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qinitauto

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qinitauto** option initializes uninitialized automatic variables to a specific value for debugging purposes.

In IBM Open XL C/C++ for AIX 17.1.1, the **-ftrivial-auto-var-init=pattern** option can be used to improve the reproducibility of issues that are caused by using uninitialized variables. However, unlike **-qinitauto**, you cannot specify the value to be assigned to an uninitialized automatic variable.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qinline

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, when you specified the **-qinline=noauto** option, only the following types of functions were considered for inlining:

- Functions that are defined with the inline specifier
- Small functions that are identified by the compiler

To achieve the same effect to inline explicitly or implicitly hinted functions, specify **-finline-hint-functions** in IBM Open XL C/C++ for AIX 17.1.1.

In IBM XL C/C++ for AIX 16.1.0 and earlier releases, **-qinline+<*function_name*>** and **-qinline-<*function_name*>** controlled automatic inlining for individual functions. In IBM Open XL C/C++ for AIX 17.1.1, there is not a functionally equivalent option. To achieve the same effect, mark the functions with the `always_inline` or `noinline` attribute.

Notes:

- The **-finline-functions** and **-finline-hint-functions** take effect only at **-O1** or higher.
- The effect of **-finline-hint-functions** is not cumulative. For example, if the option comes after **-finline-functions**, **-finline-functions** is overridden, and only explicitly or implicitly hinted functions get inlined.
- The **-fno-inline** option is overridden if it is specified together with **-finline-functions** or **-finline-hint-function**.
- The **-fno-inline-hint-functions** option is not supported by IBM Open XL C/C++ for AIX 17.1.1.

Related information

- GCC online documentation
- The "Clang command line argument reference" section in the Clang documentation

-qisolated_call

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qisolated_call`** option specifies functions in the source file that have no side effects other than those implied by their parameters.

IBM Open XL C/C++ for AIX 17.1.1 does not support a functionally equivalent option to **`-qisolated_call`**. Mark functions with the `__attribute__((pure))` function attribute instead.

Related information

- "#pragma isolated_call" on page 38
- GCC online documentation

-qkeepinlines (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qkeepinlines`** option keeps or discards definitions for unreferenced `extern` inline functions.

IBM Open XL C/C++ for AIX 17.1.1 does not support a functionally equivalent option to **`-qkeepinlines`**. Use the `__attribute__((__used__))` function attribute as an alternative. For templates, you are recommended to use C++11 explicit template instantiations.

Related information

- GCC online documentation

-qkeepparm

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qkeepparm`** option specifies whether procedure parameters are stored on the stack when used with **`-O2`** or higher optimization level.

IBM Open XL C/C++ for AIX 17.1.1 does not support a functionally equivalent option to **`-qkeepparm`**. A workaround is to compile your program without optimization enabled.

-qlargepage

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qlargepage`** option takes advantage of large pages for applications that are designed to execute in a large page memory environment.

IBM Open XL C/C++ for AIX 17.1.1 does not have an option that is functionally equivalent to **`-qlargepage`**. To make your application use large pages, large pages must be configured on the system and you must link the application with **`-Wl,-blpdata`**. See Large pages in the AIX operating system documentation for details.

-qlonglong

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qlonglong`** option allows IBM `long long` integer types in your program.

In IBM Open XL C/C++ for AIX 17.1.1, the `long long` type is allowed by default.

-qmakedep

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qmakedep`** option produces the dependency files that are used by the **`make`** tool for each source file.

In IBM Open XL C/C++ for AIX 17.1.1, the **`-M`** family of options achieve the similar effect of **`-qmakedep`**.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qminimaltoc

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qminimaltoc** option ensures that the compiler creates only one TOC entry for each compilation unit. Specifying this option can minimize the use of available TOC entries. In addition, the compiler used **-qpic=large** to support larger TOCs on AIX.

The method of using **-qminimaltoc** to reduce the number of TOC entries on AIX is no longer supported in IBM Open XL C/C++ for AIX 17.1.1. To let programs support large TOCs, specify the **-mcmodel=large** and **-Wl,-bbigtoc** options in IBM Open XL C/C++ for AIX 17.1.1.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qnamemangling (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qnamemangling** option chooses the name mangling scheme for external symbol names that are generated from C++ source code.

In IBM Open XL C/C++ for AIX 17.1.1, the Itanium-based C++ABI of **ibm-clang++** does not support prior C++ABI name mangling.

Related information

- “#pragma namemangling (C++ only), #pragma namemanglingrule (C++ only)” on page 40

-qobjmodel (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qobjmodel** option sets the object model to be used for structures, unions, and classes.

In IBM Open XL C/C++ for AIX 17.1.1, the Itanium-based C++ABI of **ibm-clang++** does not support the prior C++ABI object models.

Related information

- “#pragma object_model (C++ only)” on page 40

-qoptimize

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qoptimize** option specifies whether to optimize code during compilation and, if so, at which level.

If you were using **-qoptimize** in IBM XL C/C++ for AIX 16.1.0 or earlier releases, use **--optimize=<*level*>**, **-O0**, **-O1**, **-O2**, **-O2**, or **-Ofast** in IBM Open XL C/C++ for AIX 17.1.1 to achieve the same effect.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qpdf1, -qpdf2, -qshowpdf

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qpdf1**, **-qpdf2**, and **-qshowpdf** options tune optimizations through profile-directed feedback (PDF).

IBM Open XL C/C++ for AIX 17.1.1 no longer support the PDF feature. Profile-guided optimization (PGO) is a replacement, which is a compiler optimization technique that uses profiling to improve program runtime performance. For details, see Profile Guided Optimization (PGO).

-qppline

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-q[no]ppline** option enables or disables the generation of `#line` directives when used with the **-E** or **-P** option.

In IBM Open XL C/C++ for AIX 17.1.1, use **-E -P** to suppress emitting `#line` directives to achieve a similar effect to **-qnoppline**.

Note: The **-P** option in IBM Open XL C/C++ for AIX 17.1.1 has a completely different meaning from the **-P** option IBM XL C/C++ for AIX 16.1.0. In IBM Open XL C/C++ for AIX 17.1.1, the behavior of **-P** is consistent with that of GCC, which disables line markers in the preprocessed output for the compiler.

Related information

- "-P" on page 20
- GCC online documentation

-qprefetch

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qprefetch** option inserts prefetch instructions automatically where there are opportunities to improve code performance.

IBM Open XL C/C++ for AIX 17.1.1 does not provide functionally equivalent options to **-qprefetch=[no]assistthread** and **-qprefetch=[no]aggressive**. In IBM Open XL C/C++ for AIX 17.1.1, specify the **-mllvm -ppc-set-dscr=<*n*>** option to set the Data Stream Control Register (DSCR). For an LTO build, specify **-Wl,-bplugin_opt:--ppc-set-dscr=<*n*>** on the link step.

Related information

- **-mllvm**
- "Link Time Optimization (LTO)" on page 61

-qpriority (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qpriority** option specifies the priority level for the initialization of static objects.

IBM Open XL C/C++ for AIX 17.1.1 does not support a functionally equivalent option to **-qpriority**, nor does it support `pragma priority`. IBM Open XL C/C++ for AIX 17.1.1 supports the `__attribute__((init_priority))` attribute that provides a similar function to **-qpriority**, with which you can specify non-default priorities for initialization in source code. The default priority between IBM XL C/C++ for AIX 16.1.0 and IBM Open XL C/C++ for AIX 17.1.1 is the same. However, the value range of the priority level in `__attribute__((init_priority))` is different from that of **-qpriority**. As a result, there is no guarantee of relative ordering between the objects compiled with IBM Open XL C/C++ for AIX 17.1.1 and the objects compiled with IBM XL C/C++ for AIX 16.1.0 or earlier releases.

Related information

- "#pragma priority (C++ only)" on page 41
- GCC online documentation

-qrestrict

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, you could use the **-qrestrict** option to imply that pointer type parameters in all functions had the `restrict` keyword.

In IBM Open XL C/C++ for AIX 17.1.1, use **-frestrict-args** to achieve the same effect.

Related information

- **`-frestrict-args`**

-qsimd

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qsimd=noauto`** option disabled the conversion of loop array operations into vector instructions.

In IBM Open XL C/C++ for AIX 17.1.1, use **`-fno-vectorize`** and **`-fno-slp-vectorize`** options to achieve the same effect to disable auto vectorization features. For an LTO build, you need to add **`-Wl,-bplugin_opt:-vectorize-loops=false -Wl,-bplugin_opt:-vectorize-slp=false`** on the link step.

Related information

- "#pragma nosimd" on page 40
- The "Clang command line argument reference" section in the Clang documentation

-qsmp

IBM Open XL C/C++ for AIX 17.1.1 supports neither automatic parallelization transformations nor OpenMP. The **`-qsmp`** option is not available in IBM Open XL C/C++ for AIX 17.1.1.

-qsourcetype

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **`-qsourcetype`** option instructs the compiler to treat all recognized source files as a specified source type, regardless of the actual file name suffix. All source files following **`-qsourcetype=assembler`** are compiled as if they were assembler language source files.

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, files with either the `.S` or `.s` suffix are preprocessed before being passed to the assembler. In IBM Open XL C/C++ for AIX 17.1.1, only files with the `.S` suffix can be preprocessed. Hence, if a `.s` file needs to be preprocessed, either rename it to the `.S` file or use the **`-x assembler-with-cpp`** option to let the compiler treat it as an assembler language source file.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qstatsym

IBM Open XL C/C++ for AIX 17.1.1 produces static variables as symbols in the symbol table when the compiler does not perform optimizations. If you used **`-qstatsym`** in IBM XL C/C++ for AIX 16.1.0 or earlier releases, compile your program without optimization enabled when you migrate the program to IBM Open XL C/C++ for AIX 17.1.1.

-qstrict

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`-O3`** implies **`-qnostrict`**, which enables many floating-point optimizations. However, in IBM Open XL C/C++ for AIX 17.1.1, **`-O3`** does not enable most of the floating-point optimizations that old releases enabled. If **`-O3 -qstrict`** was used in classic XL compilers, it maps to just **`-O3`** in Open XL. To enable more floating-point optimizations in IBM Open XL C/C++ for AIX 17.1.1, try the following steps:

1. Specify **`-O3`** to let the program produce expected results.
2. Specify **`-Ofast`** to further optimize the program.
3. Customize the usage of the following options to achieve the right balance between accuracy in floating point operations and speed:

- **-f[no-]honor-infinities**
- **-f[no-]honor-nans**
- **-f[no-]math-errno**
- **-f[no-]finite-math-only**
- **-f[no-]associative-math**
- **-f[no-]reciprocal-math**
- **-f[no-]signed-zeros**
- **-f[no-]trapping-math**
- **-ffp-contract**
- **-f[no-]rounding-math**

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qtbtable

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qtbtable** option controls the amount of debugging traceback information that is included in object files.

In IBM Open XL C/C++ for AIX 17.1.1, full function traceback tables are enabled by default.

Related information

- **-mllvm**

-qtls

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qtls** option enables the recognition of the `__thread` storage class specifier and specifies the thread-local storage model to be used. The `__thread` storage class specifier designates thread-local storage for variables.

The **-ftls-model** is functionally equivalent to **-qtls**. In IBM Open XL C/C++ for AIX 17.1.1, only the **global-dynamic** suboption of this option is supported. The other tls models such as **local-dynamic** and **initial-exec** are not supported.

Note: To use thread-local storage, **-pthread** is required. **-pthread** is implied when the compiler is invoked by **ibm-clang_r** and **ibm-clang++_r**, but not implied when the compiler is invoked by **ibm-clang**.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-qtmplinst (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qtmplinst** option manages the implicit instantiation of templates.

IBM Open XL C/C++ for AIX 17.1.1 does not support a functionally equivalent option to **-qtmplinst**. You are recommended to use C++11 explicit template instantiations to control template instantiation in your program.

Related information

- GCC online documentation

-qunroll

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, automatic unrolling is enabled by the **-qunroll=auto** option at **-O2** or higher.

In IBM Open XL C/C++ for AIX 17.1.1, automatic unrolling is enabled by default at **-O2** or higher. You can also enable automatic unrolling at **-O1** by specifying the **-funroll-loops** option. In addition, **-fno-unroll-loops** can be used to disable unrolling of all loops to achieve the same effect as**-qnounroll**.

In IBM Open XL C/C++ for AIX 17.1.1, there is not an option that is functionally equivalent to **-qunroll=***n*; however, you can use **#pragma unroll(***n***)** or **#pragma clang loop unroll_count(***n***)** to control unrolling at the source level.

Related information

- #pragma unroll

-qutf

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-qutf** option enables the recognition of UTF literal syntax.

IBM Open XL C/C++ for AIX 17.1.1 does not support a functionally equivalent option to **-qutf**. UTF literal support is determined by the language level that is in effect.

-v, -V

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-v** or **-V** option reports the progress of compilation by naming the programs being invoked and the options being specified to each program.

In IBM Open XL C/C++ for AIX 17.1.1, use the **-v** option instead. Its format and compilation steps are somewhat different from those of **-v** or **-V** in the old releases.

Related information

- The "Clang command line argument reference" section in the Clang documentation

-y

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the **-y** option specifies the rounding mode for the compiler when the compiler evaluates constant floating-point expressions at compile time.

In IBM Open XL C/C++ for AIX 17.1.1, **-y** is not supported. There is no option to control the compile-time floating-point evaluation rounding mode. The Clang option **-frounding-math** gives you the ability to disable the optimizer from folding floating-point values that cannot be exactly represented; however, this might result in slower runtime performance.

The FE_TONEAREST macro, which is available in IBM Open XL C/C++ for AIX 17.1.1, matches the **-y** semantics. In IBM Open XL C/C++ for AIX 17.1.1, floating-point folding that is required by the language features such as initialization of global floating-point variables is done through FE_TONEAREST.

Related information

- "Clang Compiler User's Manual" in the Clang documentation
- The "Clang command line argument reference" section in the Clang documentation

Compiler pragmas

Consider a number of changes to compiler pragmas when you migrate your program from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1.

Supported compiler pragmas

This topic discusses compiler pragmas that are supported by IBM Open XL C/C++ for AIX 17.1.1.

Clang pragmas

Community Clang pragmas for LLVM Clang on AIX are supported in IBM Open XL C/C++ for AIX 17.1.1.

Note: Unlike IBM programs, Clang pragmas are case sensitive.

Supported IBM pragmas

The following pragmas that were supported by IBM XL C/C++ for AIX 16.1.0 are also supported by IBM Open XL C/C++ for AIX 17.1.1. Find details of these pragmas in Compiler pragmas.

- #pragma align
- #pragma GCC visibility push
- #pragma GCC visibility pop
- #pragma nosimd
- #pragma pack
- #pragma STDC FENV_ACCESS
- #pragma STDC FP_CONTRACT
- #pragma unroll
- #pragma nounroll
- #pragma unrollandfuse

  Note: #pragma nosimd and #pragma unrollandfuse are supported but deprecated in IBM Open XL C/C++ for AIX 17.1.1.

Unsupported pragmas

IBM Open XL C/C++ for AIX 17.1.1 no longer supports compiler pragmas that are described in this section.

- #pragma align(bit_packed)
- #pragma align(full)
- #pragma align(twobyte)
- #pragma align(mac68k)
- #pragma alloca (C only)
- #pragma block_loop
- #pragma chars
- #pragma comment
- #pragma complexgcc
- #pragma define
- #pragma disjoint
- #pragma do_not_instantiate (C++ only)
- #pragma enum
- #pragma execution_frequency

- #pragma expected_value
- #pragma fini (C only)
- #pragma hashome (C++ only)
- #pragma ibm independent_loop
- #pragma ibm iterations
- #pragma ibm max_iterations
- #pragma ibm min_iterations
- #pragma ibm snapshot
- #pragma implementation (C++ only)
- #pragma info
- #pragma init (C only)
- #pragma instantiate (C++ only)
- #pragma ishome (C++ only)
- #pragma isolated_call
- #pragma langlvl (C only)
- #pragma leaves
- #pragma loopid
- #pragma map
- #pragma mc_func
- #pragma namemangling (C++ only)
- #pragma namemanglingrule (C++ only)
- #pragma nofunctrace
- #pragma novector
- #pragma object_model (C++ only)
- #pragma operator_new (C++ only)
- #pragma options
- #pragma option_override
- #pragma pass_by_value (C++ only)
- #pragma priority (C++ only)
- #pragma reachable
- #pragma reg_killed_by
- #pragma report (C++ only)
- #pragma simd_level
- #pragma STDC CX_LIMITED_RANGE
- #pragma stream_unroll
- #pragma strings
- #pragma weak
- #pragma ibm independent_calls (C only)
- #pragma ibm permutation (C only)
- #pragma ibm schedule (C only)
- #pragma ibm sequential_loop (C only)
- #pragma omp atomic
- #pragma omp parallel

- #pragma omp for
- #pragma omp ordered
- #pragma omp parallel for
- #pragma omp section
- #pragma omp sections
- #pragma omp parallel sections
- #pragma omp single
- #pragma omp master
- #pragma omp critical
- #pragma omp barrier
- #pragma omp flush
- #pragma omp threadprivate
- #pragma omp task
- #pragma omp taskyield
- #pragma omp taskwait

Migration considerations of individual compiler pragmas

This section lists individual compiler pragmas that need to be considered for migration.

#pragma alloca (C only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma alloca`** is functionally equivalent to the **`-qalloca`** or **`-ma`** option. This pragma provides an inline definition of system function `alloca` when the function is called from source code that does not include the `alloca.h` header.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma alloca`**. However, you can achieve a similar function using one of the following ways:

- Specify **`-Dalloca=__builtin_alloca`** on the command line to map **`-qalloca`** to the `__builtin_alloca` function.
- Include the `alloca.h` header file in source files.

Related information

- "-qalloca, -ma (C only)" on page 21
- __builtin_alloca

#pragma chars

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma chars`** determines whether all variables of type `char` are treated as `signed` or `unsigned`.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma chars`**. However, you can use the **`-fsigned-char`** or **`-funsigned-char`** Clang option to achieve the same effect.

Related information

- The "Clang command line argument reference" section in the Clang documentation

#pragma comment

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma comment`** places a comment into object modules to indicate the program and compiler information such as the compiler version.

IBM Open XL C/C++ for AIX 17.1.1 does not support **#pragma comment**. However, when the compiler uses the AIX system assembler, the compiler version string is embedded in the object symbol table. To extract the compiler version, you can use the **dump -tv <object-file>** command or leverage other object dumpling utilities.

#pragma define (C++ only), #pragma instantiate (C++ only), #pragma do_not_instantiate (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **#pragma define**, **#pragma instantiate**, and **#pragma do_not_instantiate** provide an alternative method when explicitly instantiating a template class.

In IBM Open XL C/C++ for AIX 17.1.1, these pragmas are not supported. You are recommended to use C++11 explicit template instantiations instead.

Related information

- GCC online documentation

#pragma disjoint

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **#pragma disjoint** lists identifiers that are not aliased to each other within the scope of their use. This pragma provides more opportunities for optimizations by informing the compiler that none of the identifiers listed in the pragma shares the same physical storage.

IBM Open XL C/C++ for AIX 17.1.1 does not support **#pragma disjoint**. You are recommended to use the `restrict` type qualifier for C programs or the `__restrict__` type qualifier for C/C++ programs to assert that points are not aliased.

Related information

- GCC online documentation

#pragma enum

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **#pragma enum** is functionally equivalent to the **-qenum** option. It specifies the amount of storage occupied by enumerations.

In IBM Open XL C/C++ for AIX 17.1.1, use **-fshort-enums** as a functionally equivalent option to **-qenum=small**. Other **-qenum** suboptions are not supported. For C++ programs, you can also use the C++11 scoped enumeration feature to specify the underlying type of enumerations.

Related information

- "-qenum" on page 22
- GCC online documentation
- The "Clang command line argument reference" section in the Clang documentation

#pragma execution_frequency

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **#pragma execution_frequency** marks programs that you expect to be either very frequently or very infrequently executed.

IBM Open XL C/C++ for AIX 17.1.1 does not support **#pragma execution_frequency**. You are recommended to use the `__builtin_expect` built-in function instead.

Related information

- The __builtin_expect built-in function

#pragma expected_value

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma expected_value`** specifies the value that a parameter passed in a function call is most likely to take at run time. The compiler can use this information to perform certain optimizations, such as function cloning and inlining.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma expected_value`**. You are recommended to use the `__builtin_expect` built-in function instead.

Related information

- #__builtin_expect

#pragma fini (C only), #pragma init (C only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma init`** specifies the order in which the runtime library calls a list of functions before `main()` is called, and **`#pragma fini`** specifies the order in which the runtime library calls a list of functions after `main()` completes or `exit()` is called.

IBM Open XL C/C++ for AIX 17.1.1 does not support these pragmas. You are recommended to use the `__attribute__((constructor))` function attribute to achieve the similar function to **`#pragma init`** and use the `__attribute__((destructor))` function attribute to achieve the similar function to **`#pragma fini`**.

Related information

- GCC online documentation

#pragma GCC visibility push, #pragma GCC visibility pop

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, you could selectively set visibility attributes for entities by using pairs of the **`#pragma GCC visibility push`** and **`#pragma GCC visibility pop`** compiler directives throughout your program. **`#pragma GCC visibility push`** had the following parameters to specify visibility attributes for external linkage entities in object files:

- default
- protected
- hidden
- internal

IBM Open XL C/C++ for AIX 17.1.1 accepts and processes **`#pragma GCC visibility push`** and **`#pragma GCC visibility pop`**. However, only the **`default`** parameter of **`#pragma GCC visibility push`** is supported in IBM Open XL C/C++ for AIX 17.1.1, meaning either the visibility attribute or **`#pragma GCC visibility push`** specified in the source code is ignored by the compiler.

Related information

- #pragma GCC visibility push, #pragma GCC visibility pop

#pragma hashome (C++ only), #pragma ishome (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma ishome`** informs the compiler that the home module of the specified class is the current compilation unit. The home module is where items, such as the virtual function table, are stored. **`#pragma hashome`** informs the compiler that the specified class has a home module that is specified by **`#pragma ishome`**. The virtual function table of the specified class, along with certain inline functions, are referenced as externals in the compilation unit of the class in which **`#pragma ishome`** is specified.

IBM Open XL C/C++ for AIX 17.1.1 does not support these pragmas. Virtual function tables are emitted according to the Itanium C++ ABI rules in IBM Open XL C/C++ for AIX 17.1.1.

#pragma ibm independent_loop

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma ibm independent_loop`** explicitly states that the iterations of the chosen loop are independent and that the iterations can be executed in parallel.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma ibm independent_loop`**. You are recommended to use the **`#pragma clang loop vectorize(assume_safety)`** Clang pragma instead in the context of loop vectorization.

Related information

- "Clang Compiler User's Manual" in the Clang documentation

#pragma implementation (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, when used with the **`-qtempinc`** option, **`#pragma implementation`** supplies the name of the file that contains the template definitions corresponding to the template declarations contained in a header file.

In IBM Open XL C/C++ for AIX 17.1.1, neither **`#pragma implementation`** nor **`-qtempinc`** is supported. Use C++11 explicit template instantiations instead, which is a standard-compliant means of managing where template instantiations occur.

Related information

- GCC online documentation

#pragma info, #pragma report (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma info`** is functionally equivalent to **`-qinfo`** that produces or suppresses groups of informational messages, and **`#pragma report`** controls the generation of diagnostic messages.

IBM Open XL C/C++ for AIX 17.1.1 adopts the Clang infrastructure and has an entirely different diagnostic implementation. In IBM Open XL C/C++ for AIX 17.1.1, you can use either of the following Clang option or pragma to suppress or control the generation of diagnostic messages:

- Clang options in the form of **`-W[no-]`**
- **`#pragma clang diagnostic ignored`**

Related information

- "Diagnostic message control" on page 58
- The "Diagnostic flags in Clang" section in the Clang documentation
- The "Clang command line argument reference" section in the Clang documentation

#pragma isolated_call

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma isolated_call`** specifies functions that have no side effects in the source file other than those implied by their parameters.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma isolated_call`**. You are recommended to use the `__attribute__((pure))` function attribute instead.

Related information

- "-qisolated_call" on page 27
- GCC online documentation

#pragma langlvl (C only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma langlvl`** determines whether the source code conforms to a specific language standard, or subset or superset of a standard.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma langlvl`**. You are recommended to use the **`-std`** Clang option instead.

Related information

- The "Clang command line argument reference" section in the Clang documentation

#pragma leaves

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma leaves`** informs the compiler that a named function never returns to the instruction following a call to that function.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma leaves`**. You are recommended to use the `__attribute__((noreturn))` function attribute or **`#pragma clang attribute`** instead.

Related information

- GCC online documentation
- The "#pragma clang attribute" section in the Clang documentation

#pragma map

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma map`** converts all references to an identifier to another externally defined identifier.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma map`**. You are recommended to use the `_attribute__((alias))` function attribute or GNU `asm` labels as alternatives. For example, replace `#pragma map(foo, "bar")` with the `asm("bar")` when you migrate the following code example from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1.

Original program in IBM XL C/C++ for AIX 16.1.0

```
#pragma map(foo, "bar")
void foo();
void baz() { foo(); }
```

Migrated program in IBM Open XL C/C++ for AIX 17.1.1

```
void foo() asm("bar");
void baz() { foo(); }
```

Related information

- The "ASM Goto with Output Constraints" in the Clang documentation
- Inline assembly statements

#pragma mc_func

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma mc_func`** allows you to embed a short sequence of machine instructions "inline" within your program source code.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma mc_func`**. You are recommended to use GCC inline `asm` labels as an alternative.

Related information

- The "ASM Goto with Output Constraints" in the Clang documentation

- Inline assembly statements

#pragma namemangling (C++ only), #pragma namemanglingrule (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma namemangling`** chooses the name mangling scheme for external symbol names generated from C++ source code, and **`#pragma namemanglingrule`** provides fine-grained control over the name mangling scheme in effect for selected portions of source code, specifically with respect to the mangling of cv-qualifiers in function parameters.

In IBM Open XL C/C++ for AIX 17.1.1, neither of these pragmas is supported. IBM Open XL C/C++ for AIX 17.1.1 adopts name mangling based on the Itanium C++ ABI, so classic XL mangled names cannot be generated. This prevents you from accidentally linking objects files that are generated by IBM Open XL C/C++ for AIX 17.1.1 with objected files that are generated by IBM XL C/C++ for AIX 16.1.0 or earlier releases.

Related information

- “-qnamemangling (C++ only)” on page 28

#pragma nofunctrace

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma nofunctrace`** disables tracing for a given function or a list of specified functions.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma nofunctrace`**. You are recommended to use the `no_instrument_function` function attribute in conjunction with the **`-finstrument-functions`** option as an alternative.

Related information

- “-qfunctrace” on page 25
- GCC online documentation
- The "Clang command line argument reference" section in the Clang documentation

#pragma nosimd

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma nosimd`** disables the automatic generation of vector instructions. This pragma needs to be specified on a per-loop basis.

IBM Open XL C/C++ for AIX 17.1.1 accepts **`#pragma nosimd`** and maps it to the **`#pragma clang loop vectorize(disable)`** Clang pragma. If you used **`#pragma nosimd`** in your program, you are recommended to replace it with **`#pragma clang loop vectorize(disable)`** when you migrate the program to IBM Open XL C/C++ for AIX 17.1.1.

Related information

- “-qsimd” on page 30
- "Clang Compiler User's Manual" in the Clang documentation

#pragma object_model (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma object_model`** sets the object model to be used for structures, unions, and classes.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma object_model`**. IBM Open XL C/C++ for AIX 17.1.1 adopts the Itanium C++ ABI for object model; however, IBM XL C/C++ for AIX 16.1.0 that is invoked by **`xlC`** and earlier releases adopt a different object model. New and old object models are not compatible.

Related information

- "-qobjmodel (C++ only)" on page 28
- "Diagnostic message control" on page 58
- The "Diagnostic flags in Clang" section in the Clang documentation

#pragma operator_new (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma operator_new`** determines whether the `new` and `new[]` operators throw an exception if the requested memory cannot be allocated.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma operator_new`**. The `new` expressions in IBM Open XL C/C++ for AIX 17.1.1 are instrumented according to the C++ standard requirements, which require that a null check is instrumented if the operator `new` invoked is declared non-throwing; otherwise, a `std::bad_alloc` exception must be thrown on allocation failure.

#pragma option_override

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma option_override`** allows you to specify optimization options at the subprogram level that override optimization options given on the command line. This pragma enables finer control of program optimization and can help debug errors that occur only under optimization.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma option_override`**. If you specified **`pragma options_override(func, "opt(level, 0)")`** in your program to disable optimization for a specific function, use `__attribute__((optnone))` as an alternative when you migrate the program from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1.

Related information

- GCC online documentation

#pragma priority (C++ only)

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma priority`** specifies the priority level for the initialization of static objects.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma priority`**. However, IBM Open XL C/C++ for AIX 17.1.1 supports the `__attribute__((init_priority))` attribute that provides a similar function to **`#pragma priority`**, with which you can specify non-default priorities for static objects in source code. However, the value range of the priority level in `__attribute__((init_priority))` is different from that of **`#pragma priority`**. As a result, there is no guarantee of relative ordering between the objects compiled with IBM Open XL C/C++ for AIX 17.1.1 and the objects compiled with IBM XL C/C++ for AIX 16.1.0 or earlier releases.

Related information

- "-qpriority (C++ only)" on page 29
- GCC online documentation

#pragma reg_killed_by

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma reg_killed_by`** specifies registers that might be altered by functions that are specified by **`#pragma mc_func`**.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma mc_func`**, so **`#pragma reg_killed_by`** is not needed. If you use inline `asm` labels in place of **`#pragma mc_func`** in IBM Open XL C/C++ for AIX 17.1.1, you can use the clobber list to specify which registers are altered.

Related information

- "#pragma mc_func" on page 39
- Inline assembly statements
- The "ASM Goto with Output Constraints" in the Clang documentation

#pragma simd_level

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma simd_level`** controls the compiler code generation of vector instructions for individual loops.

When you migrate your program from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1, you can replace **`#pragma simd_level(0)`** or **`#pragma nosimd`** with **`#pragma clang loop vectorize(disabled)`** and replace **`#pragma simd_level(10)`** with **`#pragma clang loop vectorize(enable)`** respectively. IBM Open XL C/C++ for AIX 17.1.1 does not have mapping pragmas for **`#pragma simd_level`** when the simd level is from 1 to 9, inclusive.

Related information

- "Clang Compiler User's Manual" in the Clang documentation

#pragma STDC CX_LIMITED_RANGE

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma STDC CX_LIMITED_RANGE`** informs the compiler that complex division and absolute value are only invoked with values such that intermediate calculation will not overflow or lose significance.

In IBM Open XL C/C++ for AIX 17.1.1, **`#pragma STDC CX_LIMITED_RANGE`** is accepted but silently ignored.

#pragma strings

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma strings`** specifies the storage type for string literals. When **`#pragma strings(readonly)`** is in effect, strings are placed in read-only memory. When **`#pragma strings(writeable)`** is in effect, strings are placed in read-write memory.

IBM Open XL C/C++ for AIX 17.1.1 does not support **`#pragma strings`**. All strings are placed in read-only memory.

#pragma unrollandfuse

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma unrollandfuse`** instructs the compiler to attempt an unroll and fuse operation on nested `for` loops.

IBM Open XL C/C++ for AIX 17.1.1 still accepts **`#pragma unrollandfuse`** but maps it to the **`#pragma unroll_and_jam`** pragma. If you used **`#pragma unrollandfuse`** in your program, you are recommended to replace it with **`#pragma unroll_and_jam`** when you migrate the program to IBM Open XL C/C++ for AIX 17.1.1.

Related information

- "-qunroll" on page 32
- "Clang Compiler User's Manual" in the Clang documentation

#pragma weak

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, **`#pragma weak`** prevents the linker from issuing error messages if it encounters a symbol multiply-defined during linking, or if it does not find a definition for a symbol.

IBM Open XL C/C++ for AIX 17.1.1 does not support **#pragma weak**. You are recommended to use the `__attribute__((weak))` attribute or the **#pragma clang attribute push ([[weak]], apply_to = any(function))** as alternatives.

Related information

- GCC online documentation
- "Clang Compiler User's Manual" in the Clang documentation

Compiler macros

Consider a number of changes to compiler macros when you migrate your program from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1.

Supported compiler macros

This topic discusses compiler macros that are supported by IBM Open XL C/C++ for AIX 17.1.1.

IBM Open XL C/C++ for AIX 17.1.1 introduces support for new macros. Additionally, the macros that were supported by IBM XL C/C++ for AIX 16.1.0 are also supported by IBM Open XL C/C++ for AIX 17.1.1 except those described in . Find details of all supported macros in Compiler predefined macros.

Note:

IBM Open XL C/C++ for AIX 17.1.1 fully supports the C++17 language standard and predefines the **__cplusplus** macro to 201703L when the C++17 mode is enabled via the **-std=c++17** option.

Unsupported compiler macros

IBM Open XL C/C++ for AIX 17.1.1 is based on the community Clang 14.0.0 compiler. To process programs consistently with Clang, IBM Open XL C/C++ for AIX 17.1.1 no longer supports compiler macros that are described in this section.

- **Macros to identify the XL C/C++ compiler**
  - __IBMC__
  - __IBMCPP__
  - __ibmxl__
  - __ibmxl_modification__
  - __ibmxl_ptf_fix_level__
  - __ibmxl_release__
  - __ibmxl_version__
  - __ibmxl_vrm__
  - __xlc__
  - __xlC__
  - __xlC_ver__
- **Macros related to compiler option settings**
  - __DEBUG_ALLOC__
  - __IBM_DFP__
  - __IBM_DFP_SW_EMULATION__
  - __IBM_GCC_ASM
  - __IBM_STDCPP_ASM
  - __IBM_UTF_LITERAL
  - __IGNERRNO__

  - __INITAUTO__
  - __INITAUTO_W__
  - __LIBANSI__
  - __LONGDOUBLE128
  - __NO_RTTI__
  - __OBJECT_MODEL_CLASSIC__
  - __OBJECT_MODEL_IBM__
  - __RTTI_ALL__
  - __TEMPINC__
  - _CHAR_SIGNED, __CHAR_SIGNED__
  - _CHAR_UNSIGNED
  - _IBMSMP

  Note: _CHAR_UNSIGNED_ remains defined in IBM Open XL C/C++ for AIX 17.1.1.
- **Macros related to architecture settings**
  - _ARCH_COM
  - _ARCH_PPC64GR
  - _ARCH_PPC64GRSQ
  - _ARCH_PPC64V
  - _ARCH_PPC970
  - _ARCH_PWR6E
  - _ARCH_PWR6X
- **Macros related to language levels**
  - __BOOL__
  - __C99__FUNC__
  - __C99_BOOL
  - __C99_COMPLEX
  - __C99_COMPLEX_HEADER__
  - __C99_COMPOUND_LITERAL
  - __C99_CPLUSCMT
  - __C99_DESIGNATED_INITIALIZER
  - __C99_DUP_TYPE_QUALIFIER
  - __C99_EMPTY_MACRO_ARGUMENTS
  - __C99_FLEXIBLE_ARRAY_MEMBER
  - __C99_HEX_FLOAT_CONST
  - __C99_INLINE
  - __C99_LLONG
  - __C99_MACRO_WITH_VA_ARGS
  - __C99_MAX_LINE_NUMBER
  - __C99_MIXED_DECL_AND_CODE
  - __C99_MIXED_STRING_CONCAT
  - __C99_NON_CONST_AGGR_INITIALIZER
  - __C99_NON_LVALUE_ARRAY_SUB
  - __C99_PRAGMA_OPERATOR

- __C99_REQUIRE_FUNC_DECL
- __C99_RESTRICT
- __C99_STATIC_ARRAY_SIZE
- __C99_STD_PRAGMAS
- __C99_TGMATH
- __C99_UCN
- __C99_VAR_LEN_ARRAY
- __C99_VARIABLE_LENGTH_ARRAY
- __DIGRAPHS__
- __EXTENDED__
- __IBM__ALIGN
- __IBM__ALIGNOF__
- __IBM_ALIGNOF__
- __IBM_ATTRIBUTES
- __IBM_COMPUTED_GOTO
- __IBM_DOLLAR_IN_ID
- __IBM_EXTENSION_KEYWORD
- __IBM_GCC__INLINE__
- __IBM_GENERALIZED_LVALUE
- __IBM_INCLUDE_NEXT
- __IBM_LABEL_VALUE
- __IBM_LOCAL_LABEL
- __IBM_MACRO_WITH_VA_ARGS
- __IBM_NESTED_FUNCTION
- __IBM_PP_PREDICATE
- __IBM_PP_WARNING
- __IBM_REGISTER_VARS
- __IBM__TYPEOF__
- __IBMC_COMPLEX_INIT
- __IBMC_GENERIC
- __IBMC_NORETURN
- __IBMC_STATIC_ASSERT
- __IBMCPP_AUTO_TYPEDEDUCTION
- __IBMCPP_C99_LONG_LONG
- __IBMCPP_C99_PREPROCESSOR
- __IBMCPP_COMPLEX_INIT
- __IBMCPP_CONSTEXPR
- __IBMCPP_DECLTYPE
- __IBMCPP_DEFAULTED_AND_DELETED_FUNCTIONS
- __IBMCPP_DELEGATING_CTORS
- __IBMCPP_EXPLICIT_CONVERSION_OPERATORS
- __IBMCPP_EXTENDED_FRIEND
- __IBMCPP_EXTERN_TEMPLATE

- __IBMCPP_INLINE_NAMESPACE
- __IBMCPP_NULLPTR
- __IBMCPP_REFERENCE_COLLAPSING
- __IBMCPP_RIGHT_ANGLE_BRACKET
- __IBMCPP_RVALUE_REFERENCES
- __IBMCPP_SCOPED_ENUM
- __IBMCPP_STATIC_ASSERT
- __IBMCPP_VARIADIC_TEMPLATES
- __SAA__
- __SAA_L2__

The `__ibmxl` family macros are no longer supported. Instead, the `__open_xl` family macros are newly added to identify the Open XL C/C++ compiler.

Using macros to query the support of individual language features is no longer supported. You need to write source code to target language standards.

Changed compiler macros

The values of some compiler macros are changed in IBM Open XL C/C++ for AIX 17.1.1.

In IBM XL C/C++ for AIX 16.1.0, the __OPTIMIZE__ macro has the following predefined values:

- 2 when the optimization level is **-O** or **-O2**
- 3 when the optimization is **-O3**, **-O4**, or **-O5**

In IBM Open XL C/C++ for AIX 17.1.1, the predefined value of __OPTIMIZE__ is 1 for all optimization levels.

Compiler built-in functions

Consider a number of changes to compiler built-in functions when you migrate your program from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1.

Supported built-in functions

This topic discusses compiler built-in functions that are supported by IBM Open XL C/C++ for AIX 17.1.1.

IBM Open XL C/C++ for AIX 17.1.1 introduces support for Power10 built-in functions. Additionally, the built-in functions that were supported by IBM XL C/C++ for AIX 16.1.0 are also supported by IBM Open XL C/C++ for AIX 17.1.1 except those described in "Unsupported built-in functions" on page 46. Find details of all supported built-in functions in Compiler built-in functions.

Unsupported built-in functions

IBM Open XL C/C++ for AIX 17.1.1 no longer supports built-in functions that are described in this section.

- **Binary floating-point built-in functions**
  - __builtin_max
  - __builtin_min
  - __dfp_get_rounding_mode
  - __dfp_set_rounding_mode
  - __fadd
  - __fadds
  - __fmul

  - __fmuls
  - __max
  - __min
- **Binary-coded decimal built-in functions**
  - __builtin_bcdcopysign
  - __builtin_bcdsetsign
  - __builtin_bcdshift
  - __builtin_bcdshiftround
  - __builtin_bcdtruncate
  - __builtin_bcdunsignedshift
  - __builtin_bcdunsignedtruncate
  - __builtin_national2packed
  - __builtin_packed2national
  - __builtin_packed2zoned
  - __builtin_zoned2packed
- **Cache-related built-in functions**
  - __dcbtna
  - __partial_dcbt
  - __prefetch_by_load
  - __prefetch_by_stream
  - __protected_stream_count
  - __protected_stream_count_depth
  - __protected_stream_go
  - __protected_stream_set
  - __protected_stream_stride
  - __protected_stream_stop
  - __protected_stream_stop_all
  - __protected_store_stream_set
  - __protected_unlimited_stream_set
  - __protected_unlimited_store_stream_set
  - __transient_protected_stream_count_depth
  - __transient_unlimited_protected_stream_depth
  - __unlimited_protected_stream_depth
- **Cryptography built-in functions**
  - __vsbox
  - __vshasigmad
  - __vshasigmaw
- **Decimal floating-point built-in functions**
  - __addg6s
  - __cbcdtd
  - __cdtbcd
  - __d32_sNaN
  - __d64_sNaN

- __d128_sNaN
- __d32_qNaN
- __d64_qNaN
- __d128_qNaN
- __d64_abs
- __d128_abs
- __d64_biased_exponent
- __d128_biased_exponent
- __d64_compare_exponents
- __d128_compare_exponents
- __d64_compare_signaling
- __d128_compare_signaling
- __d64_copysign
- __d128_copysign
- __d64_insert_biased_exponent
- __d128_insert_biased_exponent
- __d64_integral
- __d128_integral
- __d64_integral_no_inexact
- __d128_integral_no_inexact
- __d64_isfinite
- __d128_isfinite
- __d64_isinf
- __d128_isinf
- __d64_isnan
- __d128_isnan
- __d64_isnormal
- __d128_isnormal
- __d64_issignaling
- __d128_issignaling
- __d64_issigned
- __d128_issigned
- __d64_issubnormal
- __d128_issubnormal
- __d64_iszero
- __d128_iszero
- __d64_nabs
- __d128_nabs
- __d64_shift_left
- __d128_shift_left
- __d64_shift_right
- __d128_shift_right
- __d64_to_gpr

  - __d128_to_gprs
  - __d64_to_long_long
  - __d128_to_long_long
  - __d64_to_long_long_rounding
  - __d128_to_long_long_rounding
  - __d64_to_signed_BCD
  - __d128_to_signed_BCD
  - __d64_to_unsigned_BCD
  - __d128_to_unsigned_BCD
  - __d64_quantize
  - __d128_quantize
  - __d64_reround
  - __d128_reround
  - __d64_same_quantum
  - __d128_same_quantum
  - __d64_test_data_class
  - __d128_test_data_class
  - __d64_test_data_group
  - __d128_test_data_group
  - __d64_test_significance
  - __d128_test_significance
  - __gpr_to_d64
  - __gprs_to_d128
  - __signed_BCD_to_d64
  - __signed_BCD_to_d128
  - __unsigned_BCD_to_d64
  - __unsigned_BCD_to_d128
- **Fixed-point built-in functions**
  - __assert1
  - __assert2
  - __imul_dbl
- **IBM SMP built-in functions (C only)**
  - __parthds
  - __usrthds
- **Synchronization and atomic built-in functions**
  - __check_lock_mp
  - __check_lockd_mp
  - __check_lock_up
  - __check_lockd_up
  - __clear_lock_mp
  - __clear_lockd_mp
  - __clear_lock_up
  - __clear_lockd_up

- __iospace_eieio
- __lqarx
- __stqcx

- **Transactional memory built-in functions**
  - __TM_abort
  - __TM_begin
  - __TM_end
  - __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_is_named_abort
  - __TM_nesting_depth
  - __TM_simple_begin
- **Vector built-in functions**
  - vec_extsbd
  - vec_extsbw
  - vec_extshd
  - vec_extshw
  - vec_extswd
  - vec_xxsldi
- **Miscellaneous built-in functions**
  - __fence
  - __mem_delay
  - __mftb

Changed built-in functions

Some built-in functions have been changed in IBM Open XL C/C++ for AIX 17.1.1.

In this release, you must include `altivec.h` to use the following built-in functions. For more information, see *IBM Open XL C/C++ User's Guide*.

- BCD add and subtract functions
- BCD test add and subtract for overflow functions
- BCD comparison functions
- BCD load and store functions
- Vector built-in functions

`vec_cntlz`
In IBM Open XL C/C++ and IBM XL C/C++ for AIX 16.1.0, the data types of the returned value are changed. Now the compiler returns the same type as the argument, instead of always returning an unsigned type.

You can refer to the following table for the differences:

Table 7. Result and argument types of different releases

Argument	**Result (classic releases before IBM XL C/C++ for AIX 16.1.0)**	**Result (IBM XL C/C++ for AIX 16.1.0 and IBM Open XL C/C++ releases)**
vector signed char	vector unsigned char	vector signed char
vector unsigned char	vector unsigned char	vector unsigned char
vector signed short	vector unsigned short	vector signed short
vector unsigned short	vector unsigned short	vector unsigned short
vector signed int	vector unsigned int	vector signed int
vector unsigned int	vector unsigned int	vector unsigned int
vector signed long long	vector unsigned long long	vector signed long long
vector unsigned long long	vector unsigned long long	vector unsigned long long

When you migrate programs to the latest release, this change might cause incompatibility. It is recommended that you change your code according to the new behavior.

For more information, see vec_cntlz in the *IBM Open XL C/C++ User's Guide*.

Mapping of built-in functions

The names of some built-in functions are different in IBM Open XL C/C++ for AIX 17.1.1 from those in IBM XL C/C++ for AIX 16.1.0.

The following table shows the mappings of these built-in functions.

Table 8. Mapping of built-in functions

Built-in functions in IBM Open XL C/C++ for AIX 17.1.1	**Built-in functions in IBM XL C/C++ for AIX 16.1.0**
`__builtin_ppc_addex`	`__addex`
`__builtin_ppc_alignx`	`__alignx`
`__builtin_alloca`	`__alloca`
`__builtin_ppc_bcdadd`	`__bcdadd`
`__builtin_ppc_bcdsub`	`__bcdsub`
`__builtin_bpermd`	`__bpermd`
`__builtin_ppc_cmpb`	`__cmpb`
`__builtin_ppc_cmpeqb`	`__cmpeqb`
`__builtin_complex`	`__cmplx`
`__builtin_complex`	`__cmplxf`
`__builtin_complex`	`__cmplxl`
`__builtin_ppc_cmprb`	`__cmprb`
`__builtin_clz`	`__cntlz4`
`__builtin_clzll`	`__cntlz8`

Table 8. Mapping of built-in functions (continued)

Built-in functions in IBM Open XL C/C++ for AIX 17.1.1	**Built-in functions in IBM XL C/C++ for AIX 16.1.0**
`__builtin_ctz`	`__cnttz4`
`__builtin_ctzll`	`__cnttz8`
`__builtin_ppc_compare_and_swap`	`__compare_and_swap`
`__builtin_ppc_compare_and_swaplp`	`__compare_and_swaplp`
`__builtin_ppc_compare_exp_uo`	`__compare_exp_uo`
`__builtin_ppc_compare_exp_lt`	`__compare_exp_lt`
`__builtin_ppc_compare_exp_eq`	`__compare_exp_eq`
`__builtin_ppc_compare_exp_gt`	`__compare_exp_gt`
`__builtin_darn`	`__darn`
`__builtin_darn_32`	`__darn_32`
`__builtin_darn_raw`	`__darn_raw`
`__builtin_dcbf`	`__dcbf`
`__builtin_ppc_dcbfl`	`__dcbfl`
`__builtin_ppc_dcbflp`	`__dcbflp`
`__builtin_ppc_dcbst`	`__dcbst`
`__builtin_ppc_dcbt`	`__dcbt`
`__builtin_ppc_dcbtst`	`__dcbtst`
`__builtin_ppc_dcbtstt`	`__dcbtstt`
`__builtin_ppc_dcbtt`	`__dcbtt`
`__builtin_ppc_dcbz`	`__dcbz`
`__builtin_divde`	`__divde`
`__builtin_divdeu`	`__divdeu`
`__builtin_divwe`	`__divwe`
`__builtin_divweu`	`__divweu`
`__builtin_ppc_eieio`	`__eieio`
`__builtin_ppc_extract_exp`	`__extract_exp`
`__builtin_ppc_extract_sig`	`__extract_sig`
`__builtin_ppc_fnabss`	`__fnabss`
`__builtin_ppc_fnabs`	`__fnabs`
`__builtin_ppc_fcfid`	`__fcfid`
`__builtin_ppc_fcfud`	`__fcfud`
`__builtin_ppc_fctid`	`__fctid`
`__builtin_ppc_fctidz`	`__fctidz`
`__builtin_ppc_fctiw`	`__fctiw`

Table 8. Mapping of built-in functions (continued)	
Built-in functions in IBM Open XL C/C++ for AIX 17.1.1	**Built-in functions in IBM XL C/C++ for AIX 16.1.0**
`__builtin_ppc_fctiwz`	`__fctiwz`
`__builtin_ppc_fctudz`	`__fctudz`
`__builtin_ppc_fctuwz`	`__fctuwz`
`__builtin_ppc_fetch_and_add`	`__fetch_and_add`
`__builtin_ppc_fetch_and_addlp`	`__fetch_and_addlp`
`__builtin_ppc_fetch_and_and`	`__fetch_and_and`
`__builtin_ppc_fetch_and_andlp`	`__fetch_and_andlp`
`__builtin_ppc_fetch_and_or`	`__fetch_and_or`
`__builtin_ppc_fetch_and_orlp`	`__fetch_and_orlp`
`__builtin_ppc_fetch_and_swap`	`__fetch_and_swap`
`__builtin_ppc_fetch_and_swaplp`	`__fetch_and_swaplp`
`__builtin_fma`	`__fmadd`
`__builtin_fmaf`	`__fmadds`
`__builtin_ppc_fmsub`	`__fmsub`
`__builtin_ppc_fmsubs`	`__fmsubs`
`__builtin_ppc_fnmadd`	`__fnmadd`
`__builtin_ppc_fnmadds`	`__fnmadds`
`__builtin_ppc_fnmsub`	`__fnmsub`
`__builtin_ppc_fnmsubs`	`__fnmsubs`
`__builtin_ppc_fre`	`__fre`
`__builtin_ppc_fres`	`__fres`
`__builtin_ppc_fric`	`__fric`
`__builtin_ppc_frim`	`__frim`
`__builtin_ppc_frims`	`__frims`
`__builtin_ppc_frin`	`__frin`
`__builtin_ppc_frins`	`__frins`
`__builtin_ppc_frip`	`__frip`
`__builtin_ppc_frips`	`__frips`
`__builtin_ppc_friz`	`__friz`
`__builtin_ppc_frizs`	`__frizs`
`__builtin_ppc_frsqrte`	`__frsqrte`
`__builtin_ppc_frsqrtes`	`__frsqrtes`
`__builtin_ppc_fsel`	`__fsel`
`__builtin_ppc_fsels`	`__fsels`

Table 8. Mapping of built-in functions (continued)

Built-in functions in IBM Open XL C/C++ for AIX 17.1.1	**Built-in functions in IBM XL C/C++ for AIX 16.1.0**
`__builtin_ppc_fsqrt`	`__fsqrt`
`__builtin_ppc_fsqrts`	`__fsqrts`
`__builtin_ppc_icbt`	`__icbt`
`__builtin_ppc_insert_exp`	`__insert_exp`
`__builtin_ppc_iospace_eieio`	`__iospace_eieio`
`__builtin_ppc_iospace_lwsync`	`__iospace_lwsync`
`__builtin_ppc_iospace_sync`	`__iospace_sync`
`__builtin_ppc_isync`	`__isync`
`__builtin_labs`	`__labs`
`__builtin_ppc_lbarx`	`__lbarx`
`__builtin_ppc_ldarx`	`__ldarx`
`__builtin_ppc_lharx`	`__lharx`
`__builtin_llabs`	`__llabs`
`__builtin_ppc_load2r`	`__load2r`
`__builtin_ppc_load4r`	`__load4r`
`__builtin_ppc_load8r`	`__load8r`
`__builtin_ppc_lwarx`	`__lwarx`
`__builtin_ppc_lwsync`	`__lwsync`
`__builtin_ppc_maddhd`	`__maddhd`
`__builtin_ppc_maddhdu`	`__maddhdu`
`__builtin_ppc_maddld`	`__maddld`
`__builtin_ppc_mfmsr`	`__mfmsr`
`__builtin_ppc_mfspr`	`__mfspr`
`__builtin_ppc_mftbu`	`__mftbu`
`__builtin_ppc_mtfsb0`	`__mtfsb0`
`__builtin_ppc_mtfsb1`	`__mtfsb1`
`__builtin_ppc_mtfsf`	`__mtfsf`
`__builtin_ppc_mtfsfi`	`__mtfsfi`
`__builtin_ppc_mtmsr`	`__mtmsr`
`__builtin_ppc_mtspr`	`__mtspr`
`__builtin_ppc_mulhd`	`__mulhd`
`__builtin_ppc_mulhdu`	`__mulhdu`
`__builtin_ppc_mulhw`	`__mulhw`
`__builtin_ppc_mulhwu`	`__mulhwu`

Table 8. Mapping of built-in functions (continued)

Built-in functions in IBM Open XL C/C++ for AIX 17.1.1	**Built-in functions in IBM XL C/C++ for AIX 16.1.0**
`__builtin_popcount`	`__popcnt4`
`__builtin_popcountll`	`__popcnt8`
`__builtin_ppc_popcntb`	`__popcntb`
`__builtin_ppc_poppar4`	`__poppar4`
`__builtin_ppc_poppar8`	`__poppar8`
`__builtin_ppc_rdlam`	`__rdlam`
`__builtin_readflm`	`__readflm`
`__builtin_ppc_rldimi`	`__rldimi`
`__builtin_ppc_rlwimi`	`__rlwimi`
`__builtin_ppc_rlwnm`	`__rlwnm`
`__builtin_rotateleft32`	`__rotatel4`
`__builtin_rotateleft64`	`__rotatel8`
`__builtin_ppc_setb`	`__setb`
`__builtin_setflm`	`__setflm`
`__builtin_setrnd`	`__setrnd`
`__builtin_ppc_stbcx`	`__stbcx`
`__builtin_ppc_stdcx`	`__stdcx`
`__builtin_ppc_stfiw`	`__stfiw`
`__builtin_ppc_sthcx`	`__sthcx`
`__builtin_ppc_store2r`	`__store2r`
`__builtin_ppc_store4r`	`__store4r`
`__builtin_ppc_store8r`	`__store8r`
`__builtin_ppc_stwcx`	`__stwcx`
`__builtin_ppc_swdiv`	`__swdiv`
`__builtin_ppc_swdivs`	`__swdivs`
`__builtin_ppc_swdiv_nochk`	`__swdiv_nochk`
`__builtin_ppc_swdivs_noch`	`__swdivs_nochk`
`__builtin_ppc_sync`	`__sync`
`__builtin_ppc_test_data_class`	`__test_data_class`
`__builtin_ppc_tdw`	`__tdw`
`__builtin_ppc_trap`	`__trap`
`__builtin_ppc_trapd`	`__trapd`
`__builtin_ppc_tw`	`__tw`
`__builtin_altivec_crypto_vcipher`	`__vcipher`

Table 8. Mapping of built-in functions (continued)

Built-in functions in IBM Open XL C/C++ for AIX 17.1.1	**Built-in functions in IBM XL C/C++ for AIX 16.1.0**
`__builtin_altivec_crypto_vcipherlast`	`__vcipherlast`
`__builtin_altivec_crypto_vncipher`	`__vncipher`
`__builtin_altivec_crypto_vncipherlast`	`__vncipherlast`
`__builtin_altivec_crypto_vpermxor`	`__vpermxor`
`__builtin_altivec_crypto_vpmsumb`	`__vpmsumb`
`__builtin_altivec_crypto_vpmsumd`	`__vpmsumd`
`__builtin_altivec_crypto_vpmsumh`	`__vpmsumh`
`__builtin_altivec_crypto_vpmsumw`	`__vpmsumw`

To compile source code that uses the old names of the built-in functions with IBM Open XL C/C++ for AIX 17.1.1, perform either of the following actions:

- Define the macros on the command line.
  For example:

  ```
  -D__alignx=__builtin_ppc_alignx
  ```

- Define the equivalent macros in the source code

Program linking

Consider changes to program linking when you migrate your program to IBM Open XL C/C++ for AIX 17.1.1.

You are not recommended to specify the **`-bcdtors:csect`** linker option for object code that is generated by IBM Open XL C/C++ for AIX 17.1.1. Otherwise, it might lead to crashes or incorrect results at run time. To avoid this issue, use **`-bcdtors:mbr`** instead on the link step.

In IBM Open XL C/C++ for AIX 17.1.1, many libraries are no longer linked implicitly, such as `libatomic` and `libm`. If operations related to these libraries are used in your program, specify the corresponding linking options explicitly, such as **`-latomic`** and **`-lm`**.

Compiler listings

In IBM XL C/C++ for AIX 16.1.0 and earlier releases, the **`-qlist`** option was used to produce a compiler listing file that includes object and constant area sections. In IBM Open XL C/C++ for AIX 17.1.1, you can use the **`ibm-llvm-objdump`** utility utility instead. This utility can be leveraged to print the contents of object files and final linked images named on the command line. The functionality of listing files is not provided in IBM Open XL C/C++ for AIX 17.1.1. Using the **`-S`** option, you can get an assembler language file for each source file.

In IBM XL C/C++ for AIX 16.1.0 and earlier releases, the **`-qreport`** option was used to show how sections of code have been optimized. Starting from IBM Open XL C/C++ for AIX 17.1.1, you can use the **`-Rpass`**, **`-Rpass-analysis`**, or **`-fsave-optimization-record`** LLVM options to get optimization reports. However, the reports have a different format from the listing files generated by IBM XL C/C++ for AIX 16.1.0 or earlier releases, and the information is also different. For details of **`-Rpass-remarks`**, refer to the "Options to Emit Optimization Reports" section in the Clang documentation.

Important: Differences in messages and listings between previous releases and IBM Open XL C/C++ for AIX 17.1.1 might impact compiler builds and tooling environments.

Related information

- LLVM remark diagnostics

Altivec compatibility

This section describes the changes in compiler diagnosis on incompatible vector element order, vector types, and vector built-in functions for IBM Open XL C/C++ for AIX 17.1.1 in comparison to IBM XL C/C++ for AIX 16.1.0 or earlier releases.

Compatibility of vector types

Compiler diagnosis on incompatible vector types

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, different vector types are incompatible. For example, if a variable of type `vector unsigned int` is assigned to another variable of type `vector signed int`, the compiler diagnoses the assignment. In IBM Open XL C/C++ for AIX 17.1.1, assignments between variables of different vector types are not diagnosed by default, which might result in programming errors. To enable the compiler diagnostic function in IBM Open XL C/C++ for AIX 17.1.1, specify the **`-fno-lax-vector-conversions`** option.

Operations on the vector bool and vector pixel types

IBM XL C/C++ for AIX 16.1.0 or earlier releases treat operations on the `vector bool` and `vector pixel` types differently. See the following two examples:

- **Example 1**

  In IBM XL C/C++ for AIX 16.1.0 or earlier releases, when a variable of the `vector pixel` or `vector bool` type is to be initialized with a scalar value, all elements of the vector variable are initialized with the scalar value. However, the community Clang compiler initializes only the first element of the vector variable with the scalar value while initializes the rest of elements with zero.

- **Example 2**

  In IBM XL C/C++ for AIX 16.1.0 or earlier releases, comparing two `vector bool` variables results in a scalar value. However, this comparison results in a vector variable in the community Clang compiler.

In IBM Open XL C/C++ for AIX 17.1.1, you can control the compiler behavior through the **`-faltivec-src-compat`** option. By default, the option value is **`-faltivec-src-compat=xl`** in IBM Open XL C/C++ for AIX 17.1.1 and the compiler behaves the same as IBM XL C/C++ for AIX 16.1.0 or earlier releases. However, you can change the compiler behavior to make it consistent with that of the community Clang compiler by specifying the **`-faltivec-src-compat=mixed`** option.

Compatibility of vector built-in functions

Built-in function arguments

The second argument of the following built-in functions needs to be a constant integer that ranges from 0 to 31, inclusive. In IBM XL C/C++ for AIX 16.1.0 or earlier releases, if out-of-range values are passed to the built-in functions, the compiler issues messages. However, in IBM Open XL C/C++ for AIX 17.1.1, when out-of-range values are input, the compiler does not issue messages but the behavior is undefined.

- `vec_ctd`
- `vec_ctf`
- `vec_cts`
- `vec_ctu`
- `vec_ctsl`
- `vec_ctul`

Some built-in functions expect the same argument types for two or more arguments. For example, the prototype of the `vec_add(a,b)` built-in function requires that argument b has the same type as argument a. However, in IBM XL C/C++ for AIX 16.1.0 or earlier releases, the compiler tolerates the mismatching types of a and b. In IBM Open XL C/C++ for AIX 17.1.1, the mismatching types are not allowed and the compiler issues an error message for it.

Built-in function prototypes

In IBM XL C/C++ for AIX 16.1.0 or earlier releases, the `vec_ctf`, `vec_cts`, and `vec_ctu` built-in functions have the following prototypes:

- `vector float vec_ctf(vector signed long long)`
- `vector float vec_ctf(vector unsigned long long)`
- `vector signed int vec_cts(vector double)`
- `vector unsigned int vec_ctu(vector double)`

However, these built-in functions have the following different prototypes in GCC, which are incompatible with those in IBM XL C/C++ for AIX 16.1.0 or earlier releases:

- `vector double vec_ctf(vector signed long long)`
- `vector double vec_ctf(vector unsigned long long)`
- `vector signed long long vec_cts(vector double)`
- `vector unsigned long long vec_ctu(vector double)`

In IBM Open XL C/C++ for AIX 17.1.1, you can control which set of built-in prototypes to use through the `__XL_COMPAT_ALTIVEC__` macro. By default, IBM Open XL C/C++ for AIX 17.1.1 defines the `__XL_COMPAT_ALTIVEC__` macro and provides built-in prototypes that are identical to those in IBM XL C/C++ for AIX 16.1.0 or earlier releases. If compatibility with GCC is required, you can undefine this macro by specifying the `-U__XL_COMPAT_ALTIVEC__` option.

Note: The community Clang compiler leaves `__XL_COMPAT_ALTIVEC__` undefined by default.

Unsupported built-in functions

IBM Open XL C/C++ for AIX 17.1.1 no longer supports the following built-ins that were supported in IBM XL C/C++ for AIX 16.1.0 or earlier releases:

- `vec_extsbd`[1]
- `vec_extsbw`[1]
- `vec_extshd`[1]
- `vec_extshw`[1]
- `vec_extswd`[1]
- `vec_xxsldi`[2]

Note:

1. In IBM Open XL C/C++ for AIX 17.1.1, use `vec_signexti` or `vec_signextll` as the replacement of the `vec_extsbd`, `vec_extsbw`, `vec_extshd`, `vec_extshw`, or `vec_extswd`.
2. `vec_xxsldi` was added in IBM XL C/C++ for AIX 16.1.0 invoked by **`xlclang`** or **`xlclang++`**. In IBM Open XL C/C++ for AIX 17.1.1, use `vec_sldw` as the replacement of `vec_xxsldi`.

Related information

- Vector processing support
- Vector built-in functions

Diagnostic message control

In IBM XL C/C++ for AIX 16.1.0 invoked by **`xlC`** or earlier releases, when the compiler encounters a programming error while you compile a C or C++ source program, it issues a diagnostic message to the standard error device or to a listing file. Diagnostic messages contain message numbers, message severity, message describing texts, and so on.

IBM Open XL C/C++ for AIX 17.1.1 adopts the Clang infrastructure. The diagnostic implementation in Clang is entirely different from that of the classic XL compilers. Clang provides an expressive set of diagnostic messages that do not have individual message numbers. In IBM Open XL C/C++ for AIX 17.1.1, use either of the following Clang option or pragma to suppress or control the generation of diagnostic messages:

- Clang options in the form of -W[no-]
- #pragma clang diagnostic ignored

Example

```
//t.c
#include <stdio.h>

void f(int x) {
   printf("%f", x);
}
```

Compile `t.c` with the following command:

```
ibm-clang t.c -c
```

The compiler issues the following warning message:

```
t.c:3:30: warning: format specifies type 'double' but the argument has type 'int' [-Wformat]
```

You can specify the **`-Wno-format`** option to suppress the warning message:

```
ibm-clang t.c -c -Wno-format
```

Then the compiler compiles the program with no warning messages issued.

Related information

- The "Diagnostic flags in Clang" section in the Clang documentation

Exception compatibility

This section describes the changes in compiler exception handling and propagation for IBM Open XL C/C++ for AIX 17.1.1 in comparison to IBM XL C/C++ for AIX 16.1.0 or earlier releases.

Exception handling

IBM Open XL C/C++ and IBM XL C/C++ for AIX 16.1.0 that is invoked by **`xlclang++`** generate C++ objects based on the Itanium C++ ABI.

IBM XL C/C++ for AIX 16.1.0 that is invoked by **`xlC`** or earlier releases generate C++ objects with a different C++ ABI. New and old C++ ABIs are not compatible.

If C++ objects with different C++ ABIs coexist in an application, there are limitations on exceptions thrown from objects compiled in one C++ ABI, and caught or unwound through functions compiled in a different C++ ABI.

- If an exception that is generated from code compiled with **`xlC`** is attempted to be caught or unwound in code compiled with **`xlclang++`** or **ibm-clang++_r**, the `std::terminate` handler is called.
- If an exception that is generated from code compiled with **`xlclang++`** or **ibm-clang++_r** is attempted to be caught in code compiled with **`xlC`**, the `std::terminate` handler is called.

Exception propagation (C only)

In IBM Open XL C/C++ for AIX, the default value of the **`-fexceptions`** option is **`-fno-exceptions`**. When compiling C functions, the compiler might assume that C++ exceptions cannot propagate out of the C functions by default unless the **`-fexceptions`** option is specified.

The default behavior of IBM XL C/C++ for AIX 16.1.0 and earlier releases is same as the **`-fexceptions`** option being enabled in IBM Open XL C/C++ for AIX. By default, C++ exceptions might propagate out of C functions.

Debug support

Use the **`-g`** option to enable debug support. In IBM XL C/C++ for AIX 16.1.0 and earlier releases, both the stabstrings and DWARF debugging information formats were supported and the default was stabstrings.

IBM Open XL C/C++ for AIX 17.1.1 supports only the DWARF debugging information format and the default DWARF version is DWARF 3.

In IBM Open XL C/C++ for AIX 17.1.1, you can use the following options to switch DWARF versions:

- **`-gdwarf-2`**
- **`-gdwarf, -gdwarf-3`**
- **`-gdwarf-4`**

IBM Open XL C/C++ for AIX 17.1.1 can generate DWARF information tuned for the following debuggers:

- The DBX debugger by using the **-gdbx** option, which is the default
- The GDB debugger by using the **-ggdb** option

Note: The current release of DBX does not support the DWARF information generated by IBM Open XL C/C++ for AIX 17.1.1. A future release of DBX is planned to add this support.

TLS migration considerations for debugging

For debugging purposes, it is recommended that you inspect Thread-Local Storage (TLS) variables by compiling helper functions that return the addresses of these variables and then calling these helper functions from the debugger.

Related information

- -g in the IBM Open XL C/C++ User's Guide
- The "Clang command line argument reference" section in the Clang documentation

Memory allocation

There might be heap memory allocation issues when you migrate your program to IBM Open XL C/C++ for AIX 17.1.1.

IBM Open XL C/C++ for AIX 17.1.1 pre-defines the `__VEC__` macro by default because the compiler supports POWER7® and higher processors. The `_ALL_SOURCE` macro is defined by the AIX system headers unless the macro is suppressed by other macros like `_XOPEN_SOURCE`. When both the `__VEC__` and `_ALL_SOURCE` macros are defined, the `malloc` and `calloc` system calls are mapped to `vec_malloc` and `vec_calloc` respectively in the AIX system header file `/usr/include/stdlib.h`.

Note: When both the `__VEC__` and `_ALL_SOURCE` macros are defined, the effect of the `_LINUX_SOURCE_COMPAT` macro on `malloc` and `calloc` system calls is ignored.

The `malloc` and `calloc` system calls give 8-byte aligned allocations, while `vec_malloc` and `vec_calloc` give 16-byte aligned allocations. After `malloc` and `calloc` are mapped to `vec_malloc` and `vec_calloc`, heap memory consumption is greatly increased if an application makes a lot of small heap allocations, which causes the application to run out of memory unexpectedly if the application is built with a certain `maxdata` value.

To fix the problem, try one of the following approaches:

1. Compile your program without the `_ALL_SOURCE` macro and call the `vec_malloc`, `vec_calloc`, or `posix_memalign` system call explicitly where 16-byte alignment is required.
2. Compile your program with the **`-mno-altivec`** option.
3. If the above approaches are not feasible, then for 32-bit applications, link the generated application with a larger **`-bmaxdata`** value to accommodate the extra space required due to `vec_malloc` and `vec_calloc`. For example, if you had originally specified **`-bmaxdata:0x80000000`**, you need to change the setting to a larger value, such as **`-bmaxdata:0xa0000000/dsa`**. The actual amount of additional memory depends on how the application allocates heap memory.
4. For an existing binary, set a new `maxdata` value using either the `LDR_CNTRL` environment variable or the `ldedit` command. For details, see Large program support.

OpenMP support

OpenMP is not supported in IBM Open XL C/C++ for AIX 17.1.1.

IBM Debugger for AIX

IBM Open XL C/C++ and IBM XL C/C++ for AIX 16.1 do not ship IBM Debugger for AIX.

Optimization and tuning compatibility

IBM Open XL C/C++ for AIX 17.1.1 has different optimization technology from IBM XL C/C++ for AIX 16.1.0 and earlier releases.

When you migrate programs from IBM XL C/C++ for AIX 16.1.0 or earlier releases to IBM Open XL C/C++ for AIX 17.1.1, specify LLVM options to utilize LLVM optimization features.

Link Time Optimization (LTO)

The Link Time Optimization (LTO) feature is supported in IBM Open XL C/C++ for AIX 17.1.1.

The LTO information generated by IBM Open XL C/C++ for AIX 17.1.1 is incompatible with the Interprocedural Analysis (IPA) information of IBM XL C/C++ for AIX 16.1.0 in the following aspects:

- The IPA information in object files that are created by IBM XL C/C++ for AIX 16.1.0 or earlier releases is silently ignored by IBM Open XL C/C++ for AIX 17.1.1.
- Object files that are created by IBM XL C/C++ for AIX 16.1.0 or earlier releases using the **`-qipa=noobject`** option contain only IPA information, so they are unusable in IBM Open XL C/C++ for AIX 17.1.1.
- By default, object files that are created by IBM Open XL C/C++ for AIX 17.1.1 using the **`-flto`** option are not readable by IBM XL C/C++ for AIX 16.1.0 or earlier releases.

To compile a program with LTO, specify the **`-flto`** option on both the compile and link steps:

- When **`-flto`** is specified on the compile step, the compiler generates LLVM IR instead of object code, in preparation for LTO on the link step.
- When **`-flto`** is specified on the link step, the compiler adds extra linker options such as **`-bplugin:`**<*path-to-libLTO.so*> when invoking the linker. The linker can handle a mixture of native XCOFF objects and LLVM IR objects, and then invokes the LTO optimizations on the LLVM IR objects via the `libLTO.so` plugin.

Attention:

- Pay attention to the system requirements when using LTO:

Table 9. System requirements of LTO

Feature	Minimum requirement for AIX 7.2	Minimum requirement for AIX 7.3
-flto	AIX 7.2 TL5 SP5[1]	AIX 7.3 TL1[1]
-ffat-lto-objects	AIX 7.2 TL5 SP5	AIX 7.3 TL1

Notes:

1. LTO is also supported on IBM AIX 7.2 TL5 SP4 or earlier, and IBM AIX 7.3 TL0 SP2 or earlier; however, note that non-default visibility symbols defined in the objects that are compiled with **-flto** are not necessarily retained to satisfy references from the objects that are compiled without **-flto** during linking.

- When both the **-flto** and **-ffat-lto-objects** options are specified, object files that are created by IBM Open XL C/C++ for AIX 17.1.1 are readable by IBM XL C/C++ for AIX 16.1.0 and earlier releases, but the LTO information embedded in the object files is silently ignored.
- In IBM Open XL C/C++ for AIX 17.1.1, the **-flto=thin** option is not supported. LTO mode can be set only to the full mode.

Detecting LTO usage in object files

LTO works by storing LLVM bit code, which describes the program, into object files at the compile step. During the link step, the LLVM bitcode is used to perform link-time optimization.

Detecting LTO enablement in non-fat LTO object files

By default, if a source file is compiled with the **-flto** option, only LLVM bitcode is stored into the object file. You can identify files built this way using the following file command, which identifies the object file as an LLVM IR bitcode file:

```
file file.o
file.o: LLVM IR bitcode
```

Detecting LTO enablement in fat LTO object files

If a source file is compiled with both the **-flto** and **-ffat-lto-objects** options, the resulting object file is an XCOFF object file that contains both native object code and LLVM bitcode. The LLVM bitcode is stored in the `.info` section of the XCOFF object file and is labeled with `LLVMLTO`. You can identify files built this way using the following file command:

```
strings -a file.o | grep 'LLVMLTO'
LLVMLTO
```

Alternatively, you can disassemble the object file and look for `LLVMLTO` in the `.info` section. Here is an example of a disassembled `.info` section that contains LTO information:

```
/opt/IBM/openxlC/17.1.1/libexec/ibm-llvm-objdump --section=.info --full-contents file.o
Contents of section .info:
0000 ffffffff 00000000 00000aa4 4c4c564d ............LLVM
0010 4c544f00 00000018 4243c0de 35140000 LTO.....BC..5...
```

Note: If the object file does not contain LTO information, the **ibm-llvm-objdump** utility either issues a warning message indicating the object file does not contain a `.info` section or displays a `.info` section that does not contain `LLVMLTO`.

Detecting LTO during linking

To detect whether LTO occurs at link time, use one of the following methods:

- Specify the **-bloadmap:<*filename*>** or **-bnoquiet** option during the link step, and look for `lto` in the output. All object files that participated in LTO will be displayed.

- Specify the **-bmap:<*filename*>** option during the link step, and look for `ld-temp.o` in the generated file. If `ld-temp.o` is found, it indicates some source files were compiled using **`-flto`**, and their corresponding object files participated in LTO.

Profile Guided Optimization (PGO)

Profile guided optimization (PGO), also known as profile-directed feedback (PDF), is a compiler optimization technique in computer programming that uses profiling to improve program runtime performance.

Important: IBM Open XL C/C++ for AIX 17.1.1 supports the following operating systems:

- IBM AIX 7.2: TL5 SP3 or later
- IBM AIX 7.3: TL0 or later

However, to use PGO, your operating system must be IBM AIX 7.2 TL5 SP4 or later, or IBM AIX 7.3 TL0 SP2 or later.

If you use PGO on IBM AIX 7.2 TL5 SP3 or earlier, or IBM AIX 7.3 TL0 SP1 or earlier, you might encounter the following errors:

- A segmentation fault when using the **`ibm-llvm-profdata`** utility:

```
PLEASE submit a bug report to https://ibm.biz/openxlcpp-support and include the crash
backtrace.
Stack dump:
0. Program arguments: /opt/IBM/openxlC/17.1.1/bin/ibm-llvm-profdata "ibm-llvm-profdata merge"
-o default.profdata
default_15853201381331839107_0.profraw Location 0x0000e944

--- End of call chain ---
Segmentation fault(coredump)
```

- Undefined symbols when linking:

```
ld: 0711-317 ERROR: Undefined symbol: __start___llvm_prf_cnts
ld: 0711-317 ERROR: Undefined symbol: __stop___llvm_prf_cnts
ld: 0711-317 ERROR: Undefined symbol: __start___llvm_prf_data
ld: 0711-317 ERROR: Undefined symbol: __stop___llvm_prf_data
ld: 0711-317 ERROR: Undefined symbol: __stop___llvm_prf_names
ld: 0711-317 ERROR: Undefined symbol: __start___llvm_prf_names
ld: 0711-317 ERROR: Undefined symbol: __stop___llvm_prf_vnds
ld: 0711-317 ERROR: Undefined symbol: __start___llvm_prf_vnds
```

- Linker errors:

```
ld: 0711-151 SEVERE ERROR: SETOPT: Invalid option name: NAMEDSECTS:ss
```

PGO is supported in IBM Open XL C/C++ for AIX 17.1.1. There are two ways to generate and use profile data. For more information on PGO, refer to the "Profile Guided Optimization" section in Clang documentation. PGO data files generated in IBM Open XL C/C++ for AIX 17.1.1 are incompatible with the PDF files of IBM XL C/C++ for AIX 16.1.0 or earlier releases.

The **`cleanpdf`**, **`showpdf`**, and **`mergepdf`** commands are replaced by the **`ibm-llvm-profdata`** utility in IBM Open XL C/C++ for AIX 17.1.1.

Example:

```
$ cat x.c
#include <stdio.h>
#include <stdlib.h>

int main( int argc, char *argv[] ) {
   long i = strtol(argv[1], NULL, 10);
   if (i > 5)
      printf( "i is bigger than 5\n");
   else
      printf( "i is <= 5\n");
```

```
    return 0;
}
```

To enable PGO instrumentation and instruct the compiler to instrument the code that is being compiled, specify the **-fprofile-generate[=<directory>]** option. If a directory is specified, the raw profile file is stored in that directory. Otherwise, it is stored in the current directory. The raw profile file is called `default_%m.profraw`.

```
$ ibm-clang x.c -Ofast -fprofile-generate
$ ./a.out
43
i is bigger than 5
$ ls
a.out  default_15822678448124319226_0.profraw  x.c
```

After the raw profile file is generated, run the **ibm-llvm-profdata** utility on the raw profile file to make it consumable by the compiler. Note that this step is necessary even when there is only one raw profile, since the merge operation also changes the file format.

```
$ ibm-llvm-profdata merge -o default.profdata default_15822678448124319226_0.profraw
$ ls
a.out  default_15822678448124319226_0.profraw  default.profdata  x.c
```

To instruct the compiler to use the instrumentation data to optimize the program, specify the **-fprofile-use[=<merge profile file path>]** option, where `merge profile file path` is the file location of the merged profile file. If `merge profile file path` is a directory or omitted, the name of the merged profile file is assumed to be `default.profdata`.

```
$ ibm-clang x.c -Ofast -fprofile-use
$
```

Uninitialized variables

IBM Open XL C/C++ for AIX 17.1.1 enables more aggressive optimizations than IBM Open XL C/C++ 17.1.0 and earlier classic releases. As a result, errors in programs, such as undefined behaviour due to the use of uninitialized variables are more likely to cause errors.

To detect occurrences of uninitialized local variables, compile your program using the **-Wuninitialized -Wsometimes-uninitialized** options; however, some occurrences might still remain undetected. To address the uninitialized local variable issue, initialize all local variables explicitly at their declarations. You can also use the **-ftrivial-auto-var-init=pattern** option to improve the reproducibility of the issue.

Related information

- Enhanced LLVM and Clang support

Changes to compiler default behaviors

The section shows the compiler default behavior changes between IBM Open XL C/C++ for AIX 17.1.1 and classic XL C/C++ compilers.

Changes to option defaults

The default values or behaviors of many options differ between IBM Open XL C/C++ for AIX 17.1.1 and IBM XL C/C++ for AIX 16.1.0. You can find detailed information in "Discrepancies for option defaults" on page 8 and "Migration considerations for individual compiler options" on page 18.

Allocation of uninitialized global variables

When you compile your program using the classic XL C/C++ compilers, uninitialized global variables are allocated in the common section of the object file by default.

The C mode of Open XL C/C++ compilers does not place uninitialized variables in the common section by default. To achieve behavior similar to the classic XL C/C++ compilers, specify the **`-fcommon`** option.

The C++ mode of Open XL C/C++ does not place uninitialized variables in the common section of the object file, even if you specify the **`-fcommon`** option.

Exception propagation (C only)

The exception propagation default behavior differs between IBM Open XL C/C++ for AIX and classic XL C/C++ for AIX compilers. Find details in "Exception compatibility" on page 59.

Chapter 2. Migrating from earlier Open XL releases

When you migrate programs from earlier Open XL releases, consider a number of changes such as ABI compatibility.

Discrepancies for option defaults

The section shows the discrepancies for compiler option defaults between IBM Open XL C/C++ for AIX 17.1.1 and earlier Open XL compilers.

Table 10. Option defaults on IBM Open XL C/C++ for AIX 17.1.1 and IBM Open XL C/C++ for AIX 17.1.0

Default on IBM Open XL C/C++ for AIX 17.1.0	Default on IBM Open XL C/C++ for AIX 17.1.1
-mllvm --enable-ppc-gen-scalar-mass=false when -mllvm is not specified	-mllvm --enable-ppc-gen-scalar-mass=true when -mllvm is not specified

Compatibility limitations of libc++

Consider compatibility limitations of `libc++` when you use IBM Open XL C/C++ for AIX 17.1.1.2 or earlier Open XL C/C++ releases.

If you use the LIBCXX_ENABLE_ASSERTIONS macro to enable related library assertions features in your program and compile your program using IBM Open XL C/C++ for AIX 17.1.1.2 or earlier Open XL C/C++ releases, the library assertions erroneously invoke the `__libcpp_assertion_handler` runtime library function. The `__libcpp_assertion_handler` function is not a part of the **`libc++`** ABI from LLVM 15 and might not be supported in future versions of C++ runtime libraries.

To avoid potential issues, re-compile your program with IBM Open XL C/C++ for AIX 17.1.1.3 or later releases to enable library assertions to invoke the `__libcpp_verbose_abort` function instead. Additionally, if you provide a replacement implementation of `__libcpp_assertion_handler` when using IBM Open XL C/C++ for AIX 17.1.1.3 or later releases, provide a replacement implementation for `__libcpp_verbose_abort` as well.

Support for visibility attributes

IBM Open XL C/C++ for AIX 17.1.1 no longer ignores visibility attributes as in IBM Open XL C/C++ for AIX 17.1.0 and the **`-fvisibility`** and **`-fvisibility-inlines-hidden`** options are provided to modify visibility at a translation unit level.

For more details, see Linking shared libraries and controlling symbol visibility in the *User's Guide*.

Chapter 3. Using 32-bit and 64-bit modes

You can use the IBM Open XL C/C++ compiler to develop either 32-bit or 64-bit applications.

To do so, specify **`-m32`** (the default) or **`-m64`**, respectively, during compilation. Alternatively, you can set the *OBJECT_MODE* environment variable to 32 or 64 at compile time. If both *OBJECT_MODE* and **`-m32`**/**`-m64`** are specified, **`-m32`**/**`-m64`** takes precedence.

However, porting existing applications from 32-bit to 64-bit mode can lead to a number of problems, mostly related to the differences in C/C++ long and pointer data type sizes and alignment between the two modes. The following table summarizes these differences.

Table 11. Size and alignment of data types in 32-bit and 64-bit modes

Data type	**32-bit mode**		**64-bit mode**	
	Size	**Alignment**	**Size**	**Alignment**
long, signed long, unsigned long	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries
pointer	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries
size_t (defined in the header file <cstddef>)	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries
ptrdiff_t (defined in the header file <cstddef>)	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries

The following sections discuss some of the common pitfalls implied by these differences, as well as recommended programming practices to help you avoid most of these issues:

- "Assigning long values" on page 69
- "Assigning pointers " on page 71
- "Aligning aggregate data" on page 71
- "Calling Fortran code" on page 72

Assigning long values

The limits of `long` type integers that are defined in the `limits.h` standard library header file are different in 32-bit and 64-bit modes, as shown in the following table.

Table 12. Constant limits of long integers in 32-bit and 64-bit modes

Symbolic constant	**Mode**	**Value**	**Hexadecimal**	**Decimal**
LONG_MIN (smallest signed long)	32-bit	$-(2^{31})$	0x80000000L	−2,147,483,648
	64-bit	$-(2^{63})$	0x8000000000000000L	−9,223,372,036,854,775,808
LONG_MAX (largest signed long)	32-bit	$2^{31}-1$	0x7FFFFFFFL	2,147,483,647
	64-bit	$2^{63}-1$	0x7FFFFFFFFFFFFFFFL	9,223,372,036,854,775,807
ULONG_MAX (largest unsigned long)	32-bit	$2^{32}-1$	0xFFFFFFFFUL	4,294,967,295
	64-bit	$2^{64}-1$	0xFFFFFFFFFFFFFFFFUL	18,446,744,073,709,551,615

These differences have the following implications:

- Assigning a `long` value to a `double` variable can cause loss of accuracy.

- Assigning constant values to `long` variables can lead to unexpected results. This issue is explored in more detail in "Assigning constant values to long variables" on page 70.
- Bit-shifting long values will produce different results, as described in "Bit-shifting long values" on page 71.
- Using `int` and `long` types interchangeably in expressions will lead to implicit conversion through promotions, demotions, assignments, and argument passing, and it can result in truncation of significant digits, sign shifting, or unexpected results, without warning. These operations can impact performance.

In situations where a `long` value can overflow when assigned to other variables or passed to functions, you must observe the following guidelines:

- Avoid implicit type conversion by using explicit type casting to change types.
- Ensure that all functions that accept or return `long` types are properly prototyped.
- Ensure that `long` type parameters can be accepted by the functions to which they are being passed.

Assigning constant values to long variables

Although type identification of constants follows explicit rules in C and C++, many programs use hexadecimal or unsuffixed constants as "typeless" variables and rely on a twos complement representation to truncate values that exceed the limits permitted on a 32-bit system.

As these large values are likely to be extended into a 64-bit `long` type in 64-bit mode, unexpected results can occur, generally at the following boundary areas:

- constant > `UINT_MAX`
- constant < `INT_MIN`
- constant > `INT_MAX`

Some examples of unexpected boundary side effects are listed in the following table.

Table 13. Unexpected boundary results of constants assigned to long types

Constant assigned to long	Equivalent value	32-bit mode	64-bit mode
−2,147,483,649	INT_MIN−1	+2,147,483,647	−2,147,483,649
+2,147,483,648	INT_MAX+1	−2,147,483,648	+2,147,483,648
+4,294,967,726	UINT_MAX+1	0	+4,294,967,296
0xFFFFFFFF	UINT_MAX	−1	+4,294,967,295
0x100000000	UINT_MAX+1	0	+4,294,967,296
0xFFFFFFFFFFFFFFFF	ULONG_MAX	−1	−1

Unsuffixed constants can lead to type ambiguities that can affect other parts of your program, such as when the results of `sizeof` operations are assigned to variables. For example, in 32-bit mode, the compiler types a number like 4294967295 (`UINT_MAX`) as an unsigned long and `sizeof` returns 4 bytes. In 64-bit mode, this same number becomes a signed long and `sizeof` returns 8 bytes. Similar problems occur when the compiler passes constants directly to functions.

You can avoid these problems by using the suffixes `L` (for long constants), `UL` (for unsigned long constants), `LL` (for long long constants), or `ULL` (for unsigned long long constants) to explicitly type all constants that have the potential of affecting assignment or expression evaluation in other parts of your program. In the example cited in the preceding paragraph, suffixing the number as 4294967295U forces the compiler to always recognize the constant as an `unsigned int` in 32-bit or 64-bit mode. These suffixes can also be applied to hexadecimal constants.

Bit-shifting long values

Left-bit-shifting long values produces different results in 32-bit and 64-bit modes.

The examples in Table 14 on page 71 show the effects of performing a bit-shift on long constants using the following code segment:

```
long l=valueL<<1;
```

Table 14. Results of bit-shifting long values

Initial value	Symbolic constant	Value after bit shift by one bit	
		32-bit mode	64-bit mode
0x7FFFFFFFL	INT_MAX	0xFFFFFFFE	0x00000000FFFFFFFE
0x80000000L	INT_MIN	0x00000000	0x0000000100000000
0xFFFFFFFFL	UINT_MAX	0xFFFFFFFE	0x00000001FFFFFFFE

In 32-bit mode, `0xFFFFFFFE` is negative. In 64-bit mode, `0x00000000FFFFFFFE` and `0x00000001FFFFFFFE` are both positive.

Assigning pointers

In 64-bit mode, pointers and `int` types are no longer of the same size.

The implications of this are as follows:

- Exchanging pointers and `int` types causes segmentation faults.
- Passing pointers to a function expecting an `int` type results in truncation.
- Functions that return a pointer but are not explicitly prototyped as such, return an `int` instead and truncate the resulting pointer, as illustrated in the following example.

  In C, the following code is valid in 32-bit mode without a prototype:

  ```
  a=(char*) calloc(25);
  ```

Without a function prototype for `calloc`, when the same code is compiled in 64-bit mode, the compiler assumes the function returns an `int`, so a is silently truncated and then sign-extended. Type casting the result does not prevent the truncation, as the address of the memory allocated by `calloc` was already truncated during the return. In this example, the best solution is to include the header file, `stdlib.h`, which contains the prototype for `calloc`. An alternative solution is to prototype the function as it is in the header file.

To avoid these types of problems, you can take the following measures:

- Prototype any functions that return a pointer, where possible by using the appropriate header file.
- Ensure that the type of parameter you are passing in a function, pointer or `int`, call matches the type expected by the function being called.
- For applications that treat pointers as an integer type, use type `long` or `unsigned long` in either 32-bit or 64-bit mode.

Aligning aggregate data

Normally, structures are aligned according to the most strictly aligned member in both 32-bit and 64-bit modes. However, since `long` types and pointers change size and alignment in 64-bit modes, the alignment of a structure's strictest member can change, resulting in changes to the alignment of the structure itself.

Structures that contain pointers or `long` types cannot be shared between 32-bit and 64-bit applications. Unions that attempt to share `long` and `int` types or overlay pointers onto `int` types can change the alignment. In general, you need to check all but the simplest structures for alignment and size dependencies.

In 64-bit mode, member values in a structure passed by value to a `va_arg` argument might not be accessed properly if the size of the structure is not a multiple of 8-bytes.

Any aggregate data written to a file in one mode cannot be correctly read in the other mode. Data exchanged with other languages has the similar problems.

Calling Fortran code

A significant number of applications use C, C++, and Fortran together by calling each other or sharing files.

It is currently easier to modify data sizes and types on the C and C++ sides than on the Fortran side of such applications. The following table lists C and C++ types and the equivalent Fortran types in the different modes.

Table 15. Equivalent C/C++ and Fortran data types

C/C++ type	**Fortran type**	
	32-bit	**64-bit**
signed int	INTEGER	INTEGER
signed long	INTEGER	INTEGER*8
pointer	TYPE(C_PTR[1])	TYPE(C_PTR)

Note:

1. C_PTR is provided by the ISO_C_BINDING intrinsic module.

Notices

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

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, MD-NC119
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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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.

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:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Trademarks

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.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Index

Product Number: 5765-J18; 5725-C72

GC28-3309-01