

IBM XL C/C++ for AIX  
16.1

*Migration Guide*



**Note**

Before using this information and the product it supports, read the information in [“Notices” on page 23.](#)

**First edition**

This edition applies to IBM® XL C/C++ for AIX® 16.1 (Program 5765-J12; 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.

© **Copyright International Business Machines Corporation 2021.**

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

---

# Contents

<b>About this document.....</b>	<b>V</b>
Who should read this document.....	v
How to use this document.....	v
Conventions.....	v
Related information.....	ix
Available help information.....	ix
Standards and specifications.....	xi
Technical support.....	xii
How to send your comments.....	xii
 <b>Chapter 1. Comparison between the XL-based and Clang-based front ends.....</b>	 <b>1</b>
 <b>Chapter 2. Migration checklist when moving from the XL-based front end to the Clang-based front end.....</b>	 <b>5</b>
 <b>Chapter 3. Migrating from earlier versions to the latest version.....</b>	 <b>13</b>
Migrating applications that use transactional memory built-in functions.....	14
 <b>Chapter 4. Compatibility considerations when mixing object files.....</b>	 <b>15</b>
 <b>Chapter 5. Resolving the compatibility issues of IPA object files.....</b>	 <b>17</b>
 <b>Chapter 6. Using 32-bit and 64-bit modes.....</b>	 <b>19</b>
Assigning long values.....	19
Assigning constant values to long variables.....	20
Bit-shifting long values.....	21
Assigning pointers .....	21
Aligning aggregate data.....	22
Calling Fortran code.....	22
 <b>Notices.....</b>	 <b>23</b>
Trademarks.....	25
 <b>Index.....</b>	 <b>27</b>



## About this document

---

This document contains migration considerations applicable to IBM XL C/C++ for AIX 16.1.

## Who should read this document

---

This document is intended for C and C++ developers who are to use IBM XL C/C++ for AIX 16.1 to compile programs that were previously compiled on different platforms, by previous releases of XL C/C++, 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.

Throughout this document, the **xlc**, **xlc++**, **xlc**, **xlclang**, and **xlclang++** invocation commands are used to describe the behavior of the compiler. You can, however, substitute other forms of the compiler invocation command if your particular environment requires it, and compiler option usage might differ.

While this document covers migration considerations applicable to IBM XL C/C++ for AIX 16.1, it does not include the following topics:

- An executive overview of new functions: see the *What's New for XL C/C++*.
- Compiler installation: see the *XL C/C++ Installation Guide*.
- Overview of XL C/C++ features: see the *Getting Started with XL C/C++*.
- Compiler options: see the *XL C/C++ Compiler Reference* for detailed information about the syntax and usage of compiler options.
- The C or C++ programming language: see the *XL C/C++ Language Reference* for information about the syntax, semantics, and IBM implementation of the C or C++ programming language.
- Programming topics: see the *XL C/C++ Optimization and Programming Guide* for detailed information about developing applications with XL C/C++, with a focus on program portability and optimization.

## Conventions

---

### Typographical conventions

The following table shows the typographical conventions used in the IBM XL C/C++ for AIX 16.1 information.

Table 1. Typographical conventions		
Typeface	Indicates	Example
<b>bold</b>	Lowercase commands, executable names, compiler options, and directives.	The compiler provides basic invocation commands, <b>xlc</b> and <b>xlc</b> ( <b>xlc++</b> ), along with several other compiler invocation commands to support various C/C++ language levels and compilation environments.

Table 1. Typographical conventions (continued)		
Typeface	Indicates	Example
<i>italics</i>	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 <i>size</i> parameter if you return more than the <i>size</i> requested.
<u>underlining</u>	The default setting of a parameter of a compiler option or directive.	nomaf   <u>maf</u>
monospace	Programming keywords and library functions, compiler builtins, examples of program code, command strings, or user-defined names.	To compile and optimize myprogram.c, enter: xlc myprogram.c -O3.

## Qualifying elements (icons)

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

















Table 2. Qualifying elements		
Icon	Short description	Meaning
 	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++ 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.
 	C11 begins / C11 ends	The text describes a feature that is introduced into standard C as part of C11.
 	C++11 begins / C++11 ends	The text describes a feature that is introduced into standard C++ as part of C++11.
 	C++14 begins / C++14 ends	The text describes a feature that is introduced into standard C++ as part of C++14.
 	IBM extension begins / IBM extension ends	The text describes a feature that is an IBM extension to the standard language specifications.
 	IBM XL C/C++ for AIX that is invoked by legacy invocations begins/ IBM XL C/C++ for AIX that is invoked by legacy invocations ends	The text describes a feature that is specific to IBM XL C/C++ for AIX that is invoked by legacy invocations, such as xlc and <b>xlc</b> . For the full list of the legacy invocations, see <a href="#">Legacy invocation commands</a> in the XL C/C++ Compiler Reference.

Table 2. Qualifying elements (continued)		
Icon	Short description	Meaning
	IBM XL C/C++ for AIX that is invoked by xlclang/xlclang++ invocations begins/	The text describes a feature that is specific to IBM XL C/C++ for AIX that is invoked by <b>xlclang</b> and <b>xlclang++</b> .
	IBM XL C/C++ for AIX that is invoked by xlclang/xlclang++ invocations ends	

## Syntax diagrams

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

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

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

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

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

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

Fragments, which are diagrams of syntactical units other than complete commands, directives, or statements, start with the |— symbol and end with the —| symbol.

- Required items are shown on the horizontal line (the main path):

►► keyword — *required\_argument* ►◄

- Optional items are shown below the main path:

►► keyword — *optional\_argument* ►◄

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

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

►► keyword — *required\_argument1* — *required\_argument2* ►◄

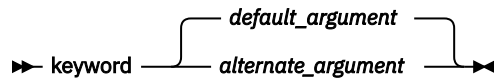
If choosing one of the items is optional, the entire stack is shown below the main path.

►► keyword — *optional\_argument1* — *optional\_argument2* ►◄

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

►► keyword — *repeatable\_argument* — ►◄

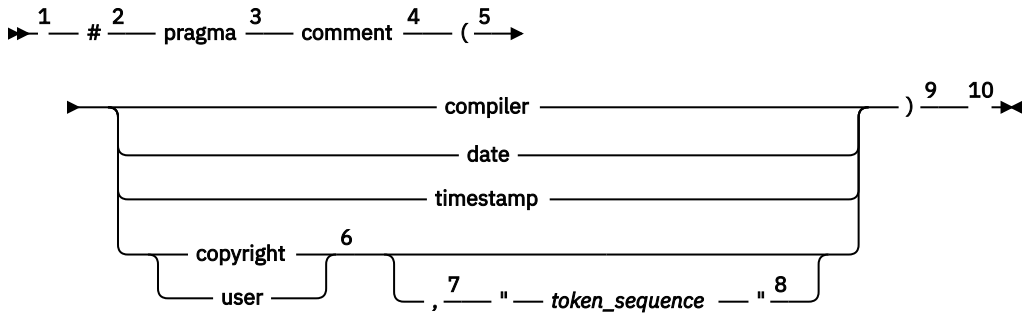
- The item that is the default is shown above the main path.



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

### Sample syntax diagram

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



Notes:

- <sup>1</sup> This is the start of the syntax diagram.
- <sup>2</sup> The symbol `#` must appear first.
- <sup>3</sup> The keyword `pragma` must appear following the `#` symbol.
- <sup>4</sup> The name of the pragma comment must appear following the keyword `pragma`.
- <sup>5</sup> An opening parenthesis must be present.
- <sup>6</sup> The comment type must be entered only as one of the types indicated: `compiler`, `date`, `timestamp`, `copyright`, or `user`.
- <sup>7</sup> A comma must appear between the comment type `copyright` or `user`, and an optional character string.
- <sup>8</sup> A character string must follow the comma. The character string must be enclosed in double quotation marks.
- <sup>9</sup> A closing parenthesis is required.
- <sup>10</sup> This is the end of the syntax diagram.

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

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

### Example of a syntax statement

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

The following list explains the syntax statement:

- Enter the keyword `EXAMPLE`.
- Enter a value for `char_constant`.
- Enter a value for `a` or `b`, but not for both.
- Optionally, enter a value for `c` or `d`.
- Enter at least one value for `e`. If you enter more than one value, you must put a comma between each.



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

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

## Examples in this information

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

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

## Related information

---

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

## Available help information

### IBM XL C/C++ information

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

- Quick Start Guide

The Quick Start Guide ([quickstart.pdf](#)) is intended to get you started with IBM XL C/C++ for AIX 16.1. It is located by default in the XL C/C++ directory and in the \quickstart directory of the installation DVD.

- README files

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

- Installable man pages

Man pages are provided for the compiler invocations and all command-line utilities provided with the product. Instructions for installing and accessing the man pages are provided in the *IBM XL C/C++ for AIX 16.1 Installation Guide*.

- Online product documentation

The fully searchable HTML-based documentation is viewable in IBM Documentation at [http://www.ibm.com/support/knowledgecenter/SSGH3R\\_16.1.0/com.ibm.compilers.aix.doc/welcome.html](http://www.ibm.com/support/knowledgecenter/SSGH3R_16.1.0/com.ibm.compilers.aix.doc/welcome.html).

- PDF documents

PDF documents are available on the web at [https://www.ibm.com/support/knowledgecenter/SSGH3R\\_16.1.0/com.ibm.compilers.aix.doc/download\\_pdf.html](https://www.ibm.com/support/knowledgecenter/SSGH3R_16.1.0/com.ibm.compilers.aix.doc/download_pdf.html).

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

**Note:** To ensure that you can access cross-reference links to other XL C/C++ 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. XL C/C++ PDF files		
Document title	PDF file name	Description
<i>What's New for IBM XL C/C++ for AIX 16.1, GC27-8053-00</i>	whats_new.pdf	Provides an executive overview of new functions in the IBM XL C/C++ for AIX 16.1 compiler, with new functions categorized according to user benefits.
<i>Getting Started with IBM XL C/C++ for AIX 16.1, SC27-8055-00</i>	getstart.pdf	Contains an introduction to XL C/C++, with information about setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
<i>IBM XL C/C++ for AIX 16.1 Installation Guide, SC27-8058-00</i>	install.pdf	Contains information for installing XL C/C++ and configuring your environment for basic compilation and program execution.
<i>IBM XL C/C++ for AIX 16.1 Migration Guide, GC27-8051-00</i>	migrate.pdf	Contains migration considerations for using XL C/C++ to compile programs that were previously compiled on different platforms, by previous releases of XL C/C++, or by other compilers.
<i>IBM XL C/C++ for AIX 16.1 Compiler Reference, SC27-8057-00</i>	compiler.pdf	Contains information about the various compiler options, pragmas, macros, environment variables, and built-in functions, including those used for parallel processing.
<i>IBM XL C/C++ for AIX 16.1 Language Reference, SC27-8059-00</i>	langref.pdf	Contains information about the C and C++ programming languages, as supported by IBM, including language extensions for portability and conformance to nonproprietary standards.
<i>IBM XL C/C++ for AIX 16.1 Optimization and Programming Guide, SC27-8060-00</i>	proguide.pdf	Contains information about advanced programming topics, such as application porting, interlanguage calls with Fortran code, library development, application optimization and parallelization, and the XL C/C++ high-performance libraries.
<i>Standard C++ Library Reference, SC27-4262-02</i>	stdlib.pdf	Contains reference information about the standard C++ runtime libraries and headers.
<i>C/C++ Legacy Class Libraries Reference, SC09-7652-00</i>	legacy.pdf	Contains reference information about the USL I/O Stream Library and the Complex Mathematics Library.

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

More information related to XL C/C++, including IBM Redbooks® publications, white papers, and other articles, is available on the web at <http://www.ibm.com/support/docview.wss?uid=swg27036618>.

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

## Other IBM information

- *Parallel Environment for AIX: Operation and Use*

- The IBM Systems Information Center, at <http://publib.boulder.ibm.com/infocenter/systems/index.jsp?topic=/com.ibm.aix.doc/doc/base/aixparent.htm>, 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*

## Other information

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

## Standards and specifications

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

- *Information Technology - Programming languages - C, ISO/IEC 9899:1990*, also known as C89.
- *Information Technology - Programming languages - C, ISO/IEC 9899:1999*, also known as C99.
- **xlc/xlc++** *Information Technology - Programming languages - C, ISO/IEC 9899:2011*, also known as C11.
- **xlc/xlc** *Information Technology - Programming languages - C, ISO/IEC 9899:2011*, also known as C11 (Partial support).
- *Information Technology - Programming languages - C++, ISO/IEC 14882:1998*, also known as C++98.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2003*, also known as C++03.
- **xlc/xlc++** *Information Technology - Programming languages - C++, ISO/IEC 14882:2011*, also known as C++11.
- **xlc/xlc** *Information Technology - Programming languages - C++, ISO/IEC 14882:2011*, also known as C++11 (Partial support).
- **xlc/xlc++** *Information Technology - Programming languages - C++, ISO/IEC 14882:2014*, also known as C++14.
- **xlc/xlc** *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/en/reference-manual/ALTIVECPIM.pdf>.
- **xlc/xlc** *Information Technology - Programming Languages - Extension for the programming language C to support decimal floating-point arithmetic, ISO/IEC WDTR 24732*. This draft technical report has been submitted to the C standards committee, and is available at <http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1176.pdf>.
- **xlc/xlc** *Decimal Types for C++: Draft 4* <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n1977.html>
- *ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985*.

## Technical support

---

Additional technical support is available from the XL C/C++ Support page at <https://www.ibm.com/mysupport/s/topic/OTO0z0000006v6TGAQ/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 [XL C, C++, and Fortran Compilers for Power servers community](https://www.ibm.com/mysupport/s/topic/OTO0z0000006v6TGAQ/xl-cc?productId=01t0z000007g72LAAQ) or open a case at <https://www.ibm.com/mysupport/s/topic/OTO0z0000006v6TGAQ/xl-cc?productId=01t0z000007g72LAAQ>.

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

## How to send your comments

---

Your feedback is important in helping us to provide accurate and high-quality information. If you have any comments or questions about this information or any other XL C/C++ information, [send compinfo@cn.ibm.com](mailto:compinfo@cn.ibm.com) an email.

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

# Chapter 1. Comparison between the XL-based and Clang-based front ends

IBM XL C/C++ for AIX 16.1 provides two front ends, which share the same back end that provides the advanced optimization technology. One front end is the IBM legacy XL-based compiler front end, which is similar with what is provided in previous releases of XL C/C++; the other is a Clang-based front end, combining the Clang front end infrastructure with the advanced optimization technology in the IBM compiler back end. For an explanation of the XL-based front end and the Clang-based front end, see [A two-in-one compiler: Clang-based front end and XL-based front end](#) in the *Getting Started with XL C/C++*.

## Overview of advantages specific to each front end and recommended user scenarios

Refer to the following table to get a rough idea on which front end to choose and its advantages. You can find detailed comparison in the rest of this topic.

Compiler front end	Advantages	Recommended user scenarios
XL-based front end	<ul style="list-style-type: none"><li>• C++ binary compatibility with releases of IBM XL C/C++ for AIX earlier than V16.1 (<a href="#">Breakage in C++ binary compatibility</a>)</li><li>• Full support for the OpenMP API Version 3.1 specification and partial support for the OpenMP API Version 4.0 specification</li></ul>	<ul style="list-style-type: none"><li>• Maintain 100% binary compatibility with legacy programs that were compiled by earlier version of IBM XL C/C++ for AIX 16.1.</li><li>• Compile code containing pragma directives compliant to the OpenMP Application Program Interface specification for explicit parallelization of C and C++ program code .</li><li>• Do not have new source code or development requirements to leverage C11, C++11, and C++14 language standard features.</li><li>• Tune your application for POWER9™ technology.</li><li>• Utilize the optimization enhancements introduced in IBM XL C/C++ for AIX.</li></ul>
Clang-based front end	<ul style="list-style-type: none"><li>• Support to the C11, C++11, and C++14 language standards (<a href="#">Different supported language levels</a>)</li><li>• Enhanced support to GCC options and pragmas (<a href="#">GCC options and pragmas available in the Clang-based front end</a>)</li></ul>	<ul style="list-style-type: none"><li>• Modernize your C++ development on AIX with more C11, C++11, and C++14 language standard features.</li><li>• Migrate from the Linux® platform or open source projects to AIX platform more easily. With the Clang-based front end, you are encouraged to use GCC options instead of legacy XL compile options where possible.</li></ul>

Compiler front end	Advantages	Recommended user scenarios
		<ul style="list-style-type: none"> <li>• Tune your application for POWER9 technology.</li> <li>• Utilize the optimization enhancements introduced in IBM XL C/C++ for AIX.</li> </ul>

## Compatibilities and incompatibilities of the Clang-based front end across platforms

- Mangled names by the Clang-based front end of IBM XL C/C++ for AIX are as defined in the Itanium C++ ABI, which is consistent with the implementation of XL C/C++ for Linux.
- The object model of the Clang-based front end of IBM XL C/C++ for AIX is based on the XL implementation of the object model defined by the Itanium C++ ABI, which the GCC object model also implements. However, objects files generated by the Clang-based front end of IBM XL C/C++ for AIX are not interoperable with the object files generated by GCC, which use different C++ standard libraries, run times, and object models, so you cannot mix and match binaries generated by **gcc** or **g++** and those by **xlclang** or **xlclang++**.

## Differences and incompatibilities between the XL-based and Clang-based front ends

### Different supported language levels

IBM XL C/C++ for AIX 16.1 modernizes its support to the C and C++ language standards by leveraging the Clang front end infrastructure. The legacy XL-based compiler front end remains unchanged in regard of supported language levels.

Table 4. Comparison of supported language levels by the XL-based and Clang-based front ends	
XL-based front end <sup>1</sup>	Clang-based front end
The compiler invoked by <b>xlc</b> provides partial support for the ISO C11 standard.	The compiler invoked by <b>xlclang</b> conforms to the ISO C11 standard with the exception of the atomics features.
The compiler invoked by <b>xlc</b> provides partial support for the ISO C++11 standard.	The compiler invoked by <b>xlclang++</b> conforms to the ISO C++11 <sup>2</sup> and C++14 standards with some dependencies on hardware and version of AIX.

### Notes:

1. Only typical invocations are listed in this table. You can refer to the [full list of compiler invocations](#) in the *XL C/C++ Compiler Reference* for all basic invocations and their equivalent special invocations.
2. Treat C++11 as a new language that is not directly interoperable with C++03 or prior language levels. For expert users, if you follow the [“Considerations when merging generated code of C++11 and C++03 or prior”](#) on page 5, C++03 or prior generated code can co-exist alongside C++11 generated code.

### Breakage in C++ binary compatibility

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, for C++ source, object files generated by the legacy XL-based front end invoked by **xlc++** are not directly interoperable with object files generated by the Clang-based front end invoked by **xlclang++**.

Although this breakage requires your additional migration efforts, the XL C/C++ compiler also takes this breakage as an opportunity to modernize other aspects of the C++ implementation.

### GCC options and pragmas available in the Clang-based front end

When using **xlclang/xlclang++** to invoke IBM XL C/C++ for AIX, you can use the GCC options and pragmas listed in the following topics. Note that these GCC options and pragmas are not available in the XL-based front end unless they were already mapped to legacy XL counterparts in releases earlier than V16.1.

- [Supported GCC options](#)
- [Supported GCC pragmas](#)

### **Unavailability of most XL legacy pragmas in the Clang-based front end**

Most IBM pragmas that are supported in releases prior to IBM XL C/C++ for AIX 16.1 are not supported in the Clang-based front end introduced in V16.1. Therefore, if you want to compile with the Clang-based front end, you need to modify your programs that were previously written for and compiled by earlier releases than IBM XL C/C++ for AIX 16.1.

For full list of unsupported pragmas, see [“Unsupported legacy XL pragmas by xlclang/xlclang++”](#) on page 9.

In the XL-based front end, the support to IBM XL pragmas remains unchanged.

## **Commonalities and compatibilities between the XL-based and Clang-based front ends**

### **C binary compatibility**

For C source, object files generated by **xlcc**, **xlcc++**, **xlclang**, and **xlclang++** are directly compatible when the symbols names in source files contain only the dollar sign and the characters from the basic character set.

You can refer to the full list of compiler invocations in the *XL C/C++ Compiler Reference* for all basic invocations and their equivalent special invocations.

### **Availability of most IBM XL legacy options**

To ease your migration, most IBM XL compiler options that are available in the XL-based front end, including the `-qoption_keyword` and flag options, are selectively supported by the Clang-based front end, which is invoked by **xlclang/xlclang++**. You are encouraged to use GCC options instead of legacy XL compile options with the Clang-based front end wherever possible.

For more information, see [Supported IBM XL compiler options by different invocations](#) in the *XL C/C++ Compiler Reference*.





## Chapter 2. Migration checklist when moving from the XL-based front end to the Clang-based front end

Aside from invoking the legacy IBM XL-based compiler front end, you can alternatively invoke the Clang-based front end provided by XL C/C++ by using the **xlclang** or **xlclang++** invocation command. As the Clang-based front end provides enhancements described in [Chapter 1, “Comparison between the XL-based and Clang-based front ends,”](#) on page 1, you might want to move to **xlclang/xlclang++**. However, you must be aware of these differences during migration.

### Binary compatibility of xlc/xlc++ and xlclang/xlclang++

Category	Case	Required action
Binary incompatibility case	For C++ source, object files generated by <b>xlc++</b> and <b>xlclang++</b> are not directly compatible as the XL-based and Clang-based front ends use different C++ standard libraries, run times, and object models.	You must recompile your legacy programs by using <b>xlclang</b> or <b>xlclang++</b> to solve such binary incompatibility and link object files successfully.
Binary compatibility case	For C source, object files generated by <b>xlc</b> , <b>xlc++</b> , <b>xlclang</b> , and <b>xlclang++</b> are directly compatible when the symbols names in source files contain only the dollar sign and the characters from the basic character set.	No action is required. You do not need to recompile source files before you link these object files.

### Considerations when merging generated code of C++11 and C++03 or prior





You should treat C++11 as a new language that is not directly interoperable with C++03 or prior language levels.

If you are an expert in the C++ library and runtime state and are willing to manage that complexity by yourself, provided you take the following considerations, C++03 or prior generated code can co-exist alongside C++11 generated code:

- There might be two C++ run times active in a single process. Modifying state in the C++ runtime, such as the `std::locale` settings, in both C++03 and C++11 modules results in undefined behavior.
- Do not delete a pointer in a C++03 module, which was allocated in a C++11 module, or vice versa.
- Do not use direct C++ language calls between C++11 and C++03 or prior. Use C interfaces instead to call between C++11 and C++03 or prior.
- Do not catch C++ exceptions thrown from C++11 in C++03 or prior, or vice versa.

### Discrepancies of option defaults between xlc/xlC and xlclang/xlclang++





Default on xlc/xlc++	Default on xlclang/xlclang++
<b>-qnortti</b>	<b>-qrtti</b>
<b>-qbitfields=unsigned</b>	Bit fields are treated as signed.
<b>-qhalt=s</b>	<b>-Werror</b> ( <b>-qhalt=w</b> ) is disabled.

Default on xlc/xlc++	Default on xlclang/xlclang++
<b>-qnothreaded</b>	<b>-qthreaded</b>
<b>-qtmplinst=auto</b>	<b>-qtmplinst=none</b>
The following legacy macros are defined: __xlc__, __xlc_ver__,  __IBMCPP__  ,  __IBMC__, and __xlc__ 	<b>-qnox1compatmacros</b> (These legacy macros are not defined.)
<b>-qvisibility=unspecified</b>	<b>-qnovisibility</b> (All the visibility pragmas and attributes that are specified in the source are ignored. )

**Note:** Only typical invocations are listed in this table. You can refer to the [full list of compiler invocations](#) in the *XL C/C++ Compiler Reference* for all basic invocations and their equivalent special invocations.

## Changed predefined macros to identify XL C/C++ when invoked by xlclang/xlclang++

For IBM XL C/C++ for AIX 16.1 that is invoked by **xlclang** or **xlclang++**, predefined macros to identify the XL C/C++ compiler are changed:

- The formerly predefined legacy macros, namely \_\_xlc\_\_, \_\_xlc\_ver\_\_,  \_\_IBMCPP\_\_ ,  \_\_IBMC\_\_, and \_\_xlc\_\_ , are no longer predefined by default.
- Some new macros are introduced and predefined by default.

For more information, see [Macros to identify the XL C/C++ compiler](#) in the *XL C/C++ Compiler Reference*.

To ease your migration efforts, option **-qx1compatmacros** is introduced, which defines the five legacy macros. Note that the compiler default for **xlclang** and **xlclang++** invocations is **-qnox1compatmacros**, meaning that the five legacy macros are not predefined. You might need to specify the **-qx1compatmacros** option when you migrate programs from IBM XL C/C++ for AIX, V13.1.3 or earlier releases to IBM XL C/C++ for AIX 16.1 that is invoked by **xlclang** or **xlclang++**.

## Unsupported legacy XL options by xlclang/xlclang++

Option	xlclang (Compiling C)	xlclang++ (Compiling C++)
-xc++	Unsupported	
-g3, -g4, -g5, -g6, -g7, -g8, -g9	Unsupported	Unsupported
-ma	Unsupported	Unsupported
-qalias=allptrs   noallptrs   global   noglobal   typeptr   notypeptr	Unsupported	Unsupported
-qalign	Unsupported	Unsupported
-qalignrulefor	Unsupported	Unsupported
-qalloca	Unsupported	Unsupported
-qasm=stdcpp	Unsupported	Unsupported
-qassert	Unsupported	Unsupported
-qattr   -qnoattr	Unsupported	Unsupported
-qbitfields	Unsupported	Unsupported
-qcinc   -qnocinc	Unsupported	Unsupported

Option	xlclang (Compiling C)	xlclang++ (Compiling C++)
-qconcurrentupdate   -qnoconcurrentupdate	Unsupported	Unsupported
-qcpluscmt   -qnocpluscmt	Unsupported	Unsupported
-qc_stdinc		Unsupported
-qcpp_stdinc	Unsupported	
-qdbcs   -qnodbcs	Unsupported	Unsupported
-qdbxextra   -qnodbxextra	Unsupported	Unsupported
-qdfp   -qnodfp	Unsupported	Unsupported
-qdigraph   -qnodigraph	Unsupported	Unsupported
-qdpcl   -qnodpcl	Unsupported	Unsupported
-qfloat=dfpemulate   nodfpemulate   fltint   nofltint   hssngl   nohssngl   rndsngl   norndsngl   single   nosingle	Unsupported	Unsupported
-qeh	Unsupported	
-qenum	Unsupported	Unsupported
-qextchk   -qnoextchk	Unsupported	Unsupported
-qflag	Unsupported	Unsupported
-qformat   -qnoformat	Unsupported	Unsupported
-qgenproto   -qnogenproto	Unsupported	Unsupported
-qhalt = i   e   s (C only) -qhalt = i   s (C++ only)	Unsupported	Unsupported
-qhaltonmsg   -qnohaltonmsg	Unsupported	Unsupported
-qheapdebug   -qnoheapdebug	Unsupported	Unsupported
-qignprag	Unsupported	Unsupported
-qinfo   -qnoinfo	Unsupported	Unsupported
-qinline=autothreshold	Unsupported	Unsupported
-qipa=threads   nothreads	Unsupported	Unsupported
-qisolated_call	Unsupported	Unsupported
-qkeepinlines	Unsupported	
-qkeyword   -qnokeyword	Unsupported	Unsupported
-qlanglvl=classic   extended   saa   saal2   <i>feature_suboption</i> (C only)  -qlanglvl=compat366   strict98   <i>feature_suboption</i> (C++ only)	Unsupported	Unsupported
-qldbl128   -qnoldbl128	Unsupported	Unsupported
-qlistfmt	Unsupported	Unsupported
-qlistopt   -qnolistopt	Unsupported	Unsupported

Option	xlclang (Compiling C)	xlclang++ (Compiling C++)
-qlongdouble   -qnolongdouble	Unsupported	Unsupported
-qlonglit   -qnolonglit	Unsupported	Unsupported
-qlonglong   -qqnolonglong	Unsupported	Unsupported
-qmacpstr   -qnomacpstr	Unsupported	Unsupported
-qmakedep	Unsupported	Unsupported
-qmaxerr   -qnomaxerr	Unsupported	Unsupported
-qmbcs   -qnombcs	Unsupported	Unsupported
-qnamemangling	Unsupported	Unsupported
-qobjmodel	Unsupported	Unsupported
-goldpassbyvalue   -qnooldpassbyvalue	Unsupported	Unsupported
-qoptdebug   -qnooptdebug	Unsupported	Unsupported
-qppline   -qnoppline	Unsupported	Unsupported
-qprint   -qnoprint	Unsupported	Unsupported
-qpriority	Unsupported	
-qproto   -qnoproto	Unsupported	Unsupported
-qrestrict   -qnorestrict	Unsupported	Unsupported
-qropt   -qnoropt	Unsupported	Unsupported
-qrtti	Unsupported	
-qshowinc   -qnoshowinc	Unsupported	Unsupported
-qshowmacros = all   pre   nopre	Unsupported	Unsupported
-qskipsrc	Unsupported	Unsupported
-qsmp   -qnosmp	Unsupported	Unsupported
-qsource   -qnosource	Unsupported	Unsupported
-qsrcmsg   -qnosrcmsg	Unsupported	Unsupported
-qstaticinline   -qnostaticinline	Unsupported	Unsupported
-qstatsym   -qnostatsym	Unsupported	Unsupported
-qsymtab	Unsupported	Unsupported
-qtabsize	Unsupported	Unsupported
-qtempinc   -qnotempinc	Unsupported	Unsupported
-qtemplatedepth	Unsupported	
-qtemplaterecompile   -qnotemplaterecompile	Unsupported	Unsupported
-qtemplateregistry   -qnotemplateregistry	Unsupported	Unsupported
-qtempmax	Unsupported	Unsupported
-qnothreaded	Unsupported	Unsupported
-qtmplinst= always   auto   noinline	Unsupported	Unsupported

Option	xlclang (Compiling C)	xlclang++ (Compiling C++)
-qtmplparse	Unsupported	Unsupported
-qtrigraph   -qnotrigraph	Unsupported	Unsupported
-qtwolink	Unsupported	
-qupconv   -qnoupconv	Unsupported	Unsupported
-qutf   -qnoutf	Unsupported	Unsupported
-qwarn0x   -qnowarn0x	Unsupported	Unsupported
-qwarn64   -qnowarn64	Unsupported	Unsupported
-qweaksymbol   -qnoweaksymbol	Unsupported	Unsupported
-qxcall   -qnoxcall	Unsupported	Unsupported
-qxref   -qnoxref	Unsupported	Unsupported

For more information, see [Supported IBM XL compiler options by different invocations](#) in the *XL C/C++ Compiler Reference*.

### Unsupported legacy XL pragmas by xlclang/xlclang++

IBM pragma	xlclang (Compiling C)	xlclang++ (Compiling C++)
#pragma alloca (C only)	Unsupported	Unsupported
#pragma block_loop	Unsupported	Unsupported
#pragma chars	Unsupported	Unsupported
#pragma comment	Unsupported	Unsupported
#pragma define (C++ only)	Unsupported	Unsupported
#pragma instantiate (C++ only)	Unsupported	Unsupported
#pragma do_not_instantiate (C++ only)	Unsupported	Unsupported
#pragma enum	Unsupported	Unsupported
#pragma expected_value	Unsupported	Unsupported
#pragma fini (C only)	Unsupported	Unsupported
#pragma hashome (C++ only)	Unsupported	Unsupported
#pragma ibm iterations	Unsupported	Unsupported
#pragma ibm max_iterations	Unsupported	Unsupported
#pragma ibm min_iterations	Unsupported	Unsupported
#pragma ibm snapshot	Unsupported	Unsupported
#pragma implementation (C++ only)	Unsupported	Unsupported
#pragma info	Unsupported	Unsupported
#pragma init (C only)	Unsupported	Unsupported
#pragma ishome (C++ only)	Unsupported	Unsupported
#pragma isolated_call	Unsupported	Unsupported

<b>IBM pragma</b>	<b>xlclang (Compiling C)</b>	<b>xlclang++ (Compiling C++)</b>
#pragma langlvl (C only)	Unsupported	Unsupported
#pragma leaves	Unsupported	Unsupported
#pragma loopid	Unsupported	Unsupported
#pragma map	Unsupported	Unsupported
#pragma mc_func	Unsupported	Unsupported
#pragma namemangling (C++ only)	Unsupported	Unsupported
#pragma namemanglingrule (C++ only)	Unsupported	Unsupported
#pragma nofunctrace	Unsupported	Unsupported
#pragma nosimd	Unsupported	Unsupported
#pragma novector	Unsupported	Unsupported
#pragma object_model (C++ only)	Unsupported	Unsupported
#pragma operator_new (C++ only)	Unsupported	Unsupported
#pragma options	Unsupported	Unsupported
#pragma pass_by_value (C++ only)	Unsupported	Unsupported
#pragma priority (C++ only)	Unsupported	Unsupported
#pragma reg_killed_by	Unsupported	Unsupported
#pragma report (C++ only)	Unsupported	Unsupported
#pragma simd_level	Unsupported	Unsupported
#pragma stream_unroll	Unsupported	Unsupported
#pragma strings	Unsupported	Unsupported
#pragma unroll	Unsupported	Unsupported
#pragma nounroll	Unsupported	Unsupported
#pragma unrollandfuse	Unsupported	Unsupported
#pragma weak	Unsupported	Unsupported
#pragma ibm independent_calls (C only)	Unsupported	Unsupported
#pragma ibm permutation (C only)	Unsupported	Unsupported
#pragma ibm schedule (C only)	Unsupported	Unsupported
#pragma ibm sequential_loop (C only)	Unsupported	Unsupported
#pragma omp atomic	Unsupported	Unsupported
#pragma omp parallel	Unsupported	Unsupported
#pragma omp for	Unsupported	Unsupported
#pragma omp ordered	Unsupported	Unsupported
#pragma omp parallel for	Unsupported	Unsupported
#pragma omp section, #pragma omp sections	Unsupported	Unsupported
#pragma omp parallel sections	Unsupported	Unsupported

IBM pragma	xlclang (Compiling C)	xlclang++ (Compiling C++)
#pragma omp single	Unsupported	Unsupported
#pragma omp master	Unsupported	Unsupported
#pragma omp critical	Unsupported	Unsupported
#pragma omp barrier	Unsupported	Unsupported
#pragma omp flush	Unsupported	Unsupported
#pragma omp threadprivate	Unsupported	Unsupported
#pragma omp task	Unsupported	Unsupported
#pragma omp taskyield	Unsupported	Unsupported
#pragma omp taskwait	Unsupported	Unsupported

For more information, see [Supported IBM pragmas by different invocations](#) in the *XL C/C++ Compiler Reference*.

### Different built-in function names between xlc/xlC and xlclang/xlclang++

The names of the built-in functions supported by IBM XL C/C++ for AIX 16.1 that is invoked by **xlclang**/**xlclang++** are in the form of `__builtin_name` while the names supported by xlc/xlC are in the form of `__name`. For example, you should use `__builtin_addex` when invoking xlclang/xlclang++ while `__addex` when invoking xlc/xlC. Note that names of the [vector built-in functions](#) are the same across two front ends.

The full list of built-in functions and their descriptions are available at [Compiler built-in functions](#) in the *XL C/C++ Compiler Reference*.

### Unsupported GCC atomic memory access built-in functions by xlclang/xlclang++

IBM XL C/C++ for AIX 16.1 that is invoked by **xlclang** or **xlclang++** does not support the following GCC atomic memory access built-in functions. Use the C++11 atomics features instead.

- Atomic fetch and operation functions
  - `__sync_fetch_and_and`
  - `__sync_fetch_and_nand`
  - `__sync_fetch_and_or`
  - `__sync_fetch_and_xor`
  - `__sync_fetch_and_add`
  - `__sync_fetch_and_sub`
- Atomic operation and fetch functions
  - `__sync_and_and_fetch`
  - `__sync_nand_and_fetch`
  - `__sync_or_and_fetch`
  - `__sync_xor_and_fetch`
  - `__sync_add_and_fetch`
  - `__sync_sub_and_fetch`
- Atomic compare and swap functions
  - `__sync_val_compare_and_swap`
  - `__sync_bool_compare_and_swap`

For more information about GCC atomic memory access built-in functions, see [GCC atomic memory access built-in functions \(IBM extension\)](#) in the *XL C/C++ Compiler Reference*.

### **Unsupported decimal floating-point built-in functions by xlclang/xlclang++**

IBM XL C/C++ for AIX 16.1 that is invoked by **xlclang** or **xlclang++** does not support decimal floating-point built-in functions.

You can find these built-in functions at [Decimal floating-point built-in functions](#) in the *XL C/C++ Compiler Reference*.

### **Changed usage for utility makeC++SharedLib and linkx1C**

If you link binaries generated by the Clang-based front end of IBM XL C/C++ for AIX by using **makeC++SharedLib** or **linkx1C**, you must also specify **-lc++** manually, in addition to the parameters you need, to make the **makeC++SharedLib** and **linkx1C** utilities work properly. If you link binaries generated by the XL-based front end, such requirement does not apply.



## Chapter 3. Migrating from earlier versions to the latest version

When you migrate programs from IBM XL C/C++ for AIX of earlier versions to the latest version, consider factors including changed compiler options, built-in functions, and environment variables.

### Changed compiler options

#### **-qaltivec**

The `altivec.h` file is no longer implicitly included when **-qaltivec** is in effect.

#### **-qslmtags**

In IBM XL C/C++ for AIX, V16.1, the option to invoke license usage tracking with SLM Tags logging is changed from **-qxflag=slmtags** to **-qslmtags**.

### Changed built-in functions

**xlcclang/xlcclang++**

In IBM XL C/C++ for AIX, V16.1, you must include `altivec.h` to use the following built-in functions. For more information, see *XL C/C++ Compiler Reference*.

- [BCD add and subtract functions](#)
- [BCD comparison functions](#)
- [BCD load and store functions](#)
- [BCD test add and subtract for overflow functions](#)
- [Vector built-in functions](#)

**xlcclang/xlcclang++**

#### **vec\_cntlz**

In IBM XL C/C++ for AIX 16.1, 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 5. Result and argument types of different releases		
Argument	Result (release versions before IBM XL C/C++ for AIX 16.1)	Result (release versions starting from IBM XL C/C++ for AIX 16.1)
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 from earlier versions to release versions starting from IBM XL C/C++ for AIX 16.1, 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 *XL C/C++ Compiler Reference*.

## Migrating applications that use transactional memory built-in functions

Starting from IBM XL C/C++ for AIX 13.1.2, to use transactional memory built-in functions, you must include a header file in the source code. In addition, if you used numeric return values of the transaction begin and end built-in functions, you must replace numeric return values with macro return values that are provided by IBM XL C/C++ for AIX 16.1.

For more information, see “[Migrating applications that use transactional memory built-in functions](#)” on page 14.

## Removed IBM Debugger for AIX

Starting from V16.1, IBM XL C/C++ for AIX does no longer ship IBM Debugger for AIX.

# Migrating applications that use transactional memory built-in functions

---

Starting from IBM XL C/C++ for AIX 13.1.2, to use transactional memory built-in functions, you must include a header file in the source code. In addition, if you used numeric return values of the transaction begin and end built-in functions, you must replace numeric return values with macro return values that are provided by IBM XL C/C++ for AIX 16.1.

## New header file needed for transactional memory built-in functions

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

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

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

### **\_\_TM\_begin**

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

### **\_\_TM\_end**

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

### **\_\_TM\_simple\_begin**

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

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

[Transactional memory built-in functions](#)

## Chapter 4. Compatibility considerations when mixing object files

### Mixing object files generated by the same compiler with different versions or option settings

Most object files that were compiled with different compilers can be linked together. However, some object files are not compatible and are restricted to be linked together. You must recompile source code to get compatible object files.

#### **xlc/xlC**

In XL C/C++ V11.1, the implementation of the threadprivate data, that is, OpenMP threadprivate variable, has been improved. The operating system thread local storage is used instead of the runtime implementation. The new implementation might improve performance on some applications.

If you plan to mix the object files .o that you have compiled with levels prior to 11.1 with the object files that you compiled with IBM XL C/C++ for AIX 16.1, and the same OpenMP threadprivate variables are referenced in both old and new object files, different implementations might cause incompatibility issues. A link error, a compile time error or other undefined behaviors might occur. To support compatibility with earlier versions, you can use the **-qsmp=noostls** suboption to switch back to the old implementation. You can recompile the entire program with the default suboption **-qsmp=ostls** to get the benefit of the new implementation.

If you are not sure whether the object files you have compiled with levels prior to 11.1 contain any old implementation, you can use the **nm** command to determine whether you need to use the **-qsmp=noostls** suboption. The following code is an example that shows how to use the **nm** command:\

```
> nm oldfiles.o
...
._xlGetThStorageBlock U      -
._xlGetThValue         U      -
...
```

In the preceding example, if `_xlGetThStorageBlock` or `_xlGetThValue` is found, this means the object files contain old implementation. In this case, you must use **-qsmp=noostls**; otherwise, use the default suboption **-qsmp=ostls**.

#### **xlc/xlC**

More compatibility considerations are as follows:

- Do not mix object and library files that were compiled with different versions of a compiler if the **-qipa** option was used during the compilation. The **-qipa** option instructs the compiler to perform an IPA link for these object and library files. An IPA link might not be able to handle mismatched versions.
- If object files were compiled with different object models, when the **-qobjmodel** option is in effect, the object files cannot be linked together and must be recompiled. The **-qobjmodel** option sets the object model to be used for structures, unions, and **C++** classes **C++**. Different object modules are not compatible.

### Mixing object files generated by the same compiler with different front ends

IBM XL C/C++ for AIX 16.1 offers two front ends, which are the legacy XL-based front end that is invoked by **xlc/xlc++** and the Clang-based front end that is invoked by **xlclang/xlclang++**. For an explanation of the XL-based front end and the Clang-based front end, see [A two-in-one compiler: Clang-based front end and XL-based front end](#) in the *Getting Started with XL C/C++*.

For C source, object files generated by **xlc**, **xlc++**, **xlclang**, and **xlclang++** are directly compatible when the symbol names in source files contain only the dollar sign or characters from the basic character

set. For C++ source, object files generated by **xlc++** and **xlclang++** are not directly compatible as the XL-based and Clang-based front ends use different C++ standard libraries, run times, and object models so that symbol mangled names are different.

Generally, symbols in object files have the following mangled names:

- Simple identifiers for C source when compiled with any of **xlc**, **xlc++**, **xlclang**, and **xlclang++**
- Simple identifiers with a `__` suffix in their names for C++ source when compiled with **xlc++**
- Identifiers with a beginning `_Z` in their names for C++ source when compiled with **xlclang++**

You can refer to the [full list of compiler invocations](#) in the *XL C/C++ Compiler Reference* for all basic invocations and their equivalent special invocations.

For example, for C function `int foo() {return 0;}`, the symbol name is `foo` when compiled with any of **xlc**, **xlc++**, **xlclang**, and **xlclang++**. For C++ function `int foo() {return 0;}`, the symbol name is `foo__Fv` when compiled with **xlc++** and `_Z3foov` when compiled with **xlclang++**.

To make binaries compatible, you are encouraged to compile source code and link the generated object files consistently with either **xlc++** or **xlclang++**.

## Mixing object files generated by the different compilers

There is no binary compatibility among AIX, Linux for big endian distributions, and Linux for little endian distributions compilers.

The objects generated by the Linux for little endian distributions compiler has a high degree of binary compatibility with objects generated by GCC.

Object files generated by IBM XL C/C++ for AIX are not interoperable with the object files generated by GCC, which use a different run time and C++ standard library, so you cannot mix and match binaries generated by GCC and IBM XL C/C++ for AIX.

**Related information** in the *XL C/C++ Migration Guide*

[“Resolving the compatibility issues of IPA object files” on page 17](#)

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

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

[-qipa](#)

[-qobjmodel](#)

**Related information** in the *XL C/C++ Optimization and Programming Guide*

[Using interprocedural analysis](#)

---

## Chapter 5. Resolving the compatibility issues of IPA object files

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

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

When IPA object files that are compiled with earlier versions of compilers are linked by a newer version, errors might occur if the IPA object is compiled by one of the following compilers.

- XL Fortran 15.1.2 or earlier
- XL C/C++ 13.1.2 or earlier

Try resolving the compatibility issue using one of the following methods:

- Recompile and link your object files with the latest XL compiler if you want to use IPA.
- Do not enable the **-qipa** option.

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

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

For more information, see [Using interprocedural analysis](#) in the *XL C/C++ Optimization and Programming Guide*.



## Chapter 6. Using 32-bit and 64-bit modes

You can use the XL C/C++ compiler to develop either 32-bit or 64-bit applications. To do so, specify **-q32** (the default) or **-q64**, respectively, during compilation. Alternatively, you can set the `OBJECT_MODE` environment variable to 32 or 64 at compile time. If both `OBJECT_MODE` and **-q32/-q64** are specified, **-q32/-q64** 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 6. 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 <stddef>)	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries
ptrdiff_t (defined in the header file <stddef>)	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 19](#)
- [“Assigning pointers” on page 21](#)
- [“Aligning aggregate data” on page 22](#)
- [“Calling Fortran code” on page 22](#)

**xlc/xlC** When compiling in 32-bit or 64-bit mode, you can use the **-qwarn64** option to help diagnose some issues related to porting applications. In either mode, the compiler immediately issues a warning if undesirable results, such as truncation or data loss, will occur when the program is executed. **xlc/xlC**

For suggestions on improving performance in 64-bit mode, see [“Optimize operations in 64-bit mode”](#) in the *XL C/C++ Optimization and Programming Guide*.

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

[-q32, -q64](#)

[-qwarn64](#)

[Compile-time and link-time environment variables](#)

## 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 7. 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

Table 7. Constant limits of long integers in 32-bit and 64-bit modes (continued)				
Symbolic constant	Mode	Value	Hexadecimal	Decimal
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 20.
- Bit-shifting long values will produce different results, as described in [“Bit-shifting long values”](#) on page 21.
- 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 8. 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 9 on page 21](#) show the effects of performing a bit-shift on long constants using the following code segment:

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

Table 9. 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	0xFFFFFFFFE	0x00000000FFFFFFFFE
0x80000000L	INT_MIN	0x00000000	0x0000000100000000
0xFFFFFFFFL	UINT_MAX	0xFFFFFFFFE	0x00000001FFFFFFFFE

In 32-bit mode, 0xFFFFFFFFE is negative. In 64-bit mode, 0x00000000FFFFFFFFE and 0x00000001FFFFFFFFE 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.
- **xlC/xlC** Use the **-qwarn64** option to get warning messages in the listing file about potential problems.

## 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.

For detailed information about aligning data structures, including structures that contain bit fields, see [Aligning data](#) in the *XL C/C++ Optimization and Programming Guide*.

## 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 10. 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
unsigned long	LOGICAL	LOGICAL*8
pointer	INTEGER	INTEGER*8
		integer POINTER (8 bytes)

## Notices

---

Programming interfaces: Intended programming interfaces allow the customer to write programs to obtain the services of IBM 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

U.S.A.

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

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

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

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

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

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

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1998, 2018.

#### PRIVACY POLICY CONSIDERATIONS:

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, or to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

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](http://www.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

## Numerics

64-bit mode  
bit-shifting [21](#)

## B

bit-shifting [21](#)









Product Number: 5765-J12; 5725-  
C72

GC27-8051-00

