

IBM XL C/C++ for Linux, V16.1



Migration Guide for Little Endian Distributions

Version 16.1

IBM XL C/C++ for Linux, V16.1



Migration Guide for Little Endian Distributions

Version 16.1

Note

Before using this information and the product it supports, read the information in “Notices” on page 19.

First edition

This edition applies to IBM XL C/C++ for Linux, V16.1 (Program 5765-J08; 5725-C73) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

© Copyright IBM Corporation 2017, 2018.

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

Contents

About this document	v
Who should read this document.	v
How to use this document.	v
Conventions	v
Related information	viii
Available help information	viii
Standards and specifications	x
Other IBM information	x
Other information	x
Technical support	xi
How to send your comments	xi

Chapter 1. Migrating from AIX to Linux for little endian distributions 1

Chapter 2. Migrating from Linux for big endian distributions to Linux for little endian distributions 3

Migrating program containing vector built-in functions from big endian systems	4
--	---

Chapter 3. Migrating from earlier versions to the latest version 5

Migrating applications that use transactional memory built-in functions	6
---	---

Chapter 4. Compatibility with GNU 9

Chapter 5. Mixing object files compiled with different compilers 11

Chapter 6. Resolving the compatibility issues of IPA object files 13

Chapter 7. Porting from 32-bit to 64-bit mode 15

Assigning long values	15
Assigning constant values to long variables.	16
Bit-shifting long values	17
Assigning pointers	17
Aligning aggregate data	18
Calling Fortran code	18

Notices 19

Trademarks	21
----------------------	----

Index 23

About this document

This document contains migration considerations applicable to IBM® XL C/C++ for Linux, V16.1.

Who should read this document

This document is intended for C and C++ developers who are to use IBM XL C/C++ for Linux, V16.1 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.”

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

While this document covers migration considerations applicable to IBM XL C/C++ for Linux, V16.1, it does not include the following topics:

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

Conventions

Typographical conventions

The following table shows the typographical conventions used in the IBM XL C/C++ for Linux, V16.1 information.

Table 1. *Typographical conventions*

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















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.
 	GPU begins / GPU ends	The text describes the information that is relevant to offloading computations to the NVIDIA GPUs.

Syntax diagrams

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

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

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

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

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

The → symbol indicates the end of a command, directive, or statement.

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

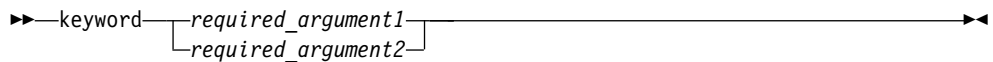
- Required items are shown on the horizontal line (the main path):



- 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.

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 Linux, V16.1. 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 and subdirectories of the installation DVD.
- Installable man pages
Man pages are provided for the compiler invocations and all command-line utilities provided with the product. Instructions for installing and accessing the man pages are provided in the *IBM XL C/C++ for Linux, V16.1 Installation Guide*.

- Online product documentation
The fully searchable HTML-based documentation is viewable in IBM Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSXVZZ_16.1.0/com.ibm.compilers.linux.doc/welcome.html.
- PDF documents
PDF documents are available on the web at https://www.ibm.com/support/knowledgecenter/SSXVZZ_16.1.0/com.ibm.compilers.linux.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 Linux, V16.1, GC27-8041-00</i>	whats_new.pdf	Provides an executive overview of new functions in the IBM XL C/C++ for Linux, V16.1 compiler, with new functions categorized according to user benefits.
<i>Getting Started with IBM XL C/C++ for Linux, V16.1, GI13-3564-00</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 Linux, V16.1 Installation Guide, GC27-8039-00</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 Linux, V16.1 Migration Guide, GC27-8042-00</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 Linux, V16.1 Compiler Reference, SC27-8047-00</i>	compiler.pdf	Contains information about the various compiler options, pragmas, macros, environment variables, and built-in functions.
<i>IBM XL C/C++ for Linux, V16.1 Language Reference, SC27-8045-00</i>	langref.pdf	Contains information about language extensions for portability and conformance to nonproprietary standards.
<i>IBM XL C/C++ for Linux, V16.1 Optimization and Programming Guide, SC27-8046-00</i>	proguide.pdf	Contains information about advanced programming topics, such as application porting, interlanguage calls with Fortran code, library development, application optimization, and the XL C/C++ high-performance libraries.

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

More information related to XL C/C++, including IBM Redbooks® publications, white papers, and other articles, is available on the web at <http://www.ibm.com/support/docview.wss?uid=swg27036675>.

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

Other IBM information

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

Other information

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

Standards and specifications

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

- *Information Technology - Programming languages - C, ISO/IEC 9899:1990*, also known as C89.
- *Information Technology - Programming languages - C, ISO/IEC 9899:1999*, also known as C99.
- *Information Technology - Programming languages - C, ISO/IEC 9899:2011*, also known as C11.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:1998*, also known as C++98.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2003*, also known as C++03.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2011*, also known as C++11.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2014*, also known as C++14 (Partial support).
- *Altivec Technology Programming Interface Manual*, Motorola Inc. This specification for vector data types, to support vector processing technology, is available at http://www.freescale.com/files/32bit/doc/ref_manual/ALTIVECPIM.pdf.
- *ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985*.
- *OpenMP Application Program Interface Version 3.1 (full support)*, *OpenMP Application Program Interface Version 4.0 (partial support)*, and *OpenMP Application Program Interface Version 4.5 (partial support)*, available at <http://www.openmp.org>

Other IBM information

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

Other information

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

Technical support

Additional technical support is available from the XL C/C++ Support page at https://www.ibm.com/support/home/product/Q833644Y89702U61/XL_C/C++_for_Linux. This page provides a portal with search capabilities to a large selection of Technotes and other support information.

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

For the latest information about XL C/C++, visit the product information site at <https://www.ibm.com/us-en/marketplace/xl-cpp-linux-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 about this information or any other XL C/C++ information, send your comments to compinfo@cn.ibm.com.

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

Chapter 1. Migrating from AIX to Linux for little endian distributions

You must consider some factors when migrating your applications from AIX® systems.

Porting from big endian systems

To help migrate programs from big endian systems, you can use the **-qaltivec=be** or **-qaltivec=le** option to toggle the vector element sequence in registers to big endian or little endian element order.

For more information, see “Migrating program containing vector built-in functions from big endian systems” on page 4.

Changed compiler options

-qxlcompatmacros

Starting from IBM XL C/C++ for Linux, V13.1.6, **-qnoxlcompatmacros** is set to default. To define these legacy macros: `__xlC__`, `__xlC_ver__`, `__cplusplus`, `__IBMCPP__`, `__C++`, `__C`, `__IBMC__`, and `__xlC__`, you must explicitly specify **-qxlcompatmacros**.

Changed built-in functions

Starting from IBM XL C/C++ for Linux, V13.1.6, you must include `altivec.h` to use the following built-in functions.

- 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

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



-qxlcompatmacros



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

Chapter 2. Migrating from Linux for big endian distributions to Linux for little endian distributions

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

Porting from big endian systems

- To help migrate programs from big endian systems, you can use the **-qaltivec=be** or **-qaltivec=le** option to toggle the vector element sequence in registers to big endian or little endian element order.

For more information, see “Migrating program containing vector built-in functions from big endian systems” on page 4.

Changed compiler options

-qxlcompatmacros

Starting from IBM XL C/C++ for Linux, V13.1.6, **-qnoxlcompatmacros** is set to default. To define these legacy macros: `__xlC__`, `__xlC_ver__`, `__cplusplus`, `__IBMCPP__`, `__IBMC__`, and `__xlC__`, you must explicitly specify **-qxlcompatmacros**.

Changed built-in functions

Starting from IBM XL C/C++ for Linux, V13.1.6, you must include `altivec.h` to use the following built-in functions.

- 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

For more information, see XL C/C++ Compiler Reference.

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



-maltivec (-qaltivec)



-qxlcompatmacros



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

Migrating program containing vector built-in functions from big endian systems

When migrating the programs that contain the Vector Multimedia Extension (VMX) and Vector Scalar Extension (VSX) built-in functions from big endian systems, you can use `-qaltivec=be` or `-maltivec=be` to minimize program changes, but you need to pay attention in specific cases.

The following table shows what users need to pay attention when migrating code from big endian systems by using `-qaltivec=be` or `-maltivec=be`.

Table 4. Attention when -qaltivec=be and -maltivec=be

Case	Attention
If the existing program contains only VMX load and store built-in functions	Using <code>-qaltivec=be</code> or <code>-maltivec=be</code> may affect the program performance; using <code>-qaltivec=le</code> or <code>-maltivec=le</code> may affect the performance in different ways.
If the existing program contains only VSX load and store built-in functions	In the existing programs, you can use the <code>vec_xl</code> and <code>vec_xst</code> functions to replace the VSX load and store built-in functions to maximally simplify the code changes.
If the existing program contains both VMX and VSX load and store built-in functions	You need to pay attention to the differences of the element order of vectors that are operated by the VMX and VSX built-in functions in little endian systems.
If the existing program contains the vector initialization by using union with arrays	You need to use the <code>vec_ld</code> or <code>vec_xl</code> function to load the vectors explicitly, instead of using the union with arrays, or you can reverse the element order of the array used for vector initialization.
Vector literals	Based on the meaning and usage of vector literals, the user must change the code properly.

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



`-qaltivec`



Supported GCC options



Vector built-in functions

Chapter 3. Migrating from earlier versions to the latest version

When you migrate applications from earlier versions to the latest version, consider factors including changed compiler options, built-in functions, and environment variables.

Changed compiler options

-qaltivec

Starting from IBM XL C/C++ for Linux, V13.1.6, the **-qaltivec** option takes effect only when you include the `altivec.h` file and set or imply **-mcpu** (**-qarch**) to be an architecture that supports vector instructions. Otherwise, the compiler ignores **-qaltivec** and issues a warning message.

For more information, see `-maltivec` (`-qaltivec`) in the *XL C/C++ Compiler Reference*.

-qxlcompatmacros

Starting from IBM XL C/C++ for Linux, V13.1.6, **-qnoxlcompatmacros** is set to default. To define these legacy macros: `__xlC__`, `__xlC_ver__`, `__cplusplus__`, `__IBMCPP__`, `__C__`, `__IBMC__`, and `__xlC__`, you must explicitly specify **-qxlcompatmacros**.

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

Changed environment variables

XLSMPOPTS

Starting from IBM XL C/C++ for Linux, V13.1.6, **XLSMPOPTS=target=optional** is renamed to **XLSMPOPTS=target=default** with the identical functionality, and **XLSMPOPTS=target=disable** is renamed to **XLSMPOPTS=target=disabled** with the identical functionality.

For more information, see `XLSMPOPTS` in the *XL C/C++ Compiler Reference*.

Changed built-in functions

Starting from IBM XL C/C++ for Linux, V13.1.6, 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

vec_cntlz

Starting from IBM XL C/C++ for Linux, V13.1.5, 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 Linux, V13.1.5)	Result (release versions starting from IBM XL C/C++ for Linux, V13.1.5)
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 Linux, V13.1.5 for little endian distributions, 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 Linux, V13.1.2, to use transactional memory built-in functions, you must include a header file in the source code. In addition, if you used numeric return values of the transaction begin and end built-in functions, you must replace numeric return values with macro return values that are provided by IBM XL C/C++ for Linux, V16.1.

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 with GNU

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

IBM XL C/C++ for Linux is built with Clang front end and IBM optimizing back end components. It provides improved GNU Compiler Collection (GCC) compatibility and language standards support for easier migration and enhanced capability as well as the IBM optimization technology. IBM XL C/C++ for Linux, V16.1 supports the use of `gcc` and `g++` compiler options and therefore the `gxc` and `gxlc++` invocation commands are not required or included.

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

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

IBM XL C/C++ for Linux, V16.1 has been fully tested with the following GNU compilers:

- GNU C/C++ 5.3.1 on Ubuntu Server 16.04
- GNU C/C++ 4.8.5 on RHEL 7.3
- GNU C/C++ 4.8.5 on RHEL 7.4
- GNU C/C++ 4.8.5 on SLES 12 SP3
- GNU C/C++ 4.8.3 on SLES 12

.

Notes: Some additional noteworthy points about this relationship are as follows:

- IBM built-in functions coexist with GNU C built-ins.
- Compilation of C and C++ programs uses the GNU C and GNU C++ header files.
- Compilation uses the GNU assembler for assembler input files.
- Compiled C code is linked to the GNU C runtime libraries.
- Compiled C++ code is linked to the GNU C and GNU C++ runtime libraries.
- Code compiled with XL C/C++ can be debugged with the GNU debugger, `gdb`.

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

- Supported GCC options
- Supported GCC pragmas
- Supported GCC built-in functions
 - GCC atomic memory access built-in functions
 - GCC object size checking built-in functions

- Supported GCC vector built-in functions
- Supported GCC non-vector built-in functions

Related reference in the *XL C/C++ Language Reference*

- Supported GNU C/C++ features
 - Supported features for C and C++
 - Supported features for C only
 - Supported features for C++ only

Chapter 5. Mixing object files compiled with different compilers

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

Note the following restrictions:

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

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



-qipa

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



Using interprocedural analysis

Chapter 6. Resolving the compatibility issues of IPA object files

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

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

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

- XL Fortran, V15.1.2 or earlier
- XL C/C++, V13.1.2 or earlier

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

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

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

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

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

Chapter 7. Porting from 32-bit to 64-bit mode

IBM XL C/C++ for Linux, V16.1 supports only 64-bit compilation mode, which means you can use the XL C/C++ compiler to develop only 64-bit applications.

You might want to port existing 32-bit applications to the 64-bit IBM XL C/C++ for Linux, V16.1. However, this 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”
- “Assigning pointers” on page 17
- “Aligning aggregate data” on page 18
- “Calling Fortran code” on page 18

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



Compile-time and link-time environment variables

Assigning long values

The limits of long type integers defined in the `limits.h` standard library header file are shown in the following table.

Table 7. Constant limits of long integers in 64-bit mode

Symbolic constant	Value	Hexadecimal	Decimal
LONG_MIN (smallest signed long)	-2^{63}	0x8000000000000000L	-9,223,372,036,854,775,808
LONG_MAX (largest signed long)	$2^{63}-1$	0x7FFFFFFFFFFFFFFFL	9,223,372,036,854,775,807

Table 7. Constant limits of long integers in 64-bit mode (continued)

Symbolic constant	Value	Hexadecimal	Decimal
ULONG_MAX (largest unsigned long)	$2^{64}-1$	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.”
- Bit-shifting long values will produce different results, as described in “Bit-shifting long values” on page 17.
- 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 two's 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

Unsuffix 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

The examples in Table 9 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
0x7FFFFFFFL	INT_MAX	0x00000000FFFFFFFE
0x80000000L	INT_MIN	0x0000000100000000
0xFFFFFFFFL	UINT_MAX	0x00000001FFFFFFFFFE

In 32-bit mode, 0xFFFFFFFF is negative. In 64-bit mode, 0x00000000FFFFFFFFFE and 0x00000001FFFFFFFFFE 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.

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.

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 Linux.

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.

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

NVIDIA is either registered trademark or trademark of NVIDIA Corporation in the United States, other countries, or both.

Index

Numerics

- 64-bit mode
 - bit-shifting 17
 - data types 15
 - long constants 16
 - pointers 17

A

- aggregate data
 - aligning 18
- alignment
 - 32-bit mode 18
 - 64-bit mode 18

B

- bit-shifting 17

C

- compatibility
 - GNU 9
- constants
 - long types 16
- customization
 - GNU 9

D

- data types
 - Fortran 18
 - long 15

F

- Fortran
 - data types
 - C/C++ 18

L

- long types
 - values 15

M

- migration
 - code 4

P

- pointers
 - 64-bit mode 17



Product Number: 5765-J08; 5725-C73

Printed in USA

GC27-8042-00

