IBM XL C/C++ for Blue Gene/Q, V12.1

Getting Started with XL C/C++

Version 12.1

GC14-7361-00

IBM XL C/C++ for Blue Gene/Q, V12.1

Getting Started with XL C/C++

Version 12.1

GC14-7361-00

First edition

This edition applies to IBM XL C/C++ for Blue Gene/Q, V12.1 (Program 5799-AG1) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Contents

About this document

This document contains overview and basic usage information for the IBM® XL C/C++ for Blue Gene®/Q, V12.1 compiler.

Who should read this document

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

How to use this document

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

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

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

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

Conventions

Typographical conventions

The following table shows the typographical conventions used in the IBM XL C/C++ for Blue Gene/Q, V12.1 information.

Table 1. Typographical conventions

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

Qualifying elements (icons)

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

Table 2. Qualifying elements

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

Table 2. Qualifying elements (continued)

Qualifier/Icon	**Meaning**
C1X, or C1X begins ▶ C1X C1X ◀ C1X ends	The text describes a feature that is introduced into standard C as part of C1X.
C++0x, or C++0x begins ▶ C++0x C++0x ◀ C++0x ends	The text describes a feature that is introduced into standard C++ as part of C++0x.

Syntax diagrams

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

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

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

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

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

  The ───▶◀ symbol indicates the end of a command, directive, or statement.

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

```
▶▶──keyword──required_argument──────────────────────────────────────────▶◀
```

- Optional items are shown below the main path:

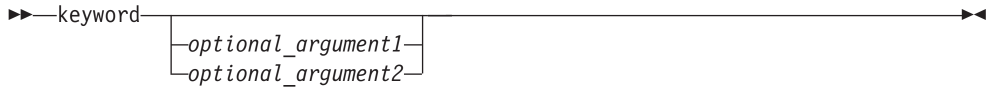
```
▶▶──keyword──┬─────────────────────┬────────────────────────────────────▶◀
             └─optional_argument───┘
```

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

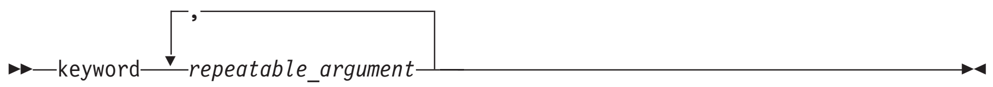
  If you *must* choose one of the items, one item of the stack is shown on the main path.

```
▶▶──keyword──┬─required_argument1─┬─────────────────────────────────────▶◀
             └─required_argument2─┘
```

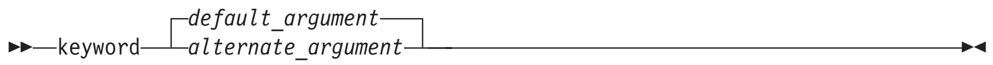
  If choosing one of the items is optional, the entire stack is shown below the main path.



- 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:



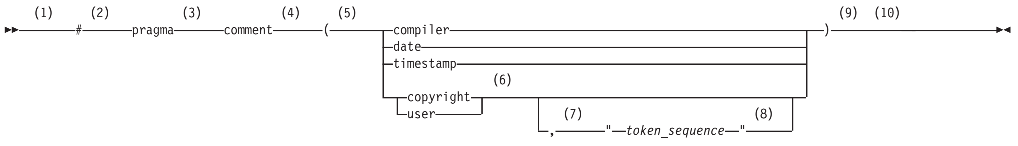
- The item that is the default is shown above the main path.



- 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.

Sample syntax diagram

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



Notes:

1 This is the start of the syntax diagram.

2 The symbol # must appear first.

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

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

5 An opening parenthesis must be present.

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

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

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

9 A closing parenthesis is required.

10 This is the end of the syntax diagram.

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

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

Examples in this information

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

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

Related information

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

IBM XL C/C++ information

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

- README files

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

  Man pages are provided for the compiler invocations and all command-line utilities provided with the product. Instructions for installing and accessing the man pages are provided in the *IBM XL C/C++ for Blue Gene/Q, V12.1 Installation Guide*.
- Information center

  The information center of searchable HTML files can be launched on a network and accessed remotely or locally. Instructions for installing and accessing the online information center are provided in the *IBM XL C/C++ for Blue Gene/Q, V12.1 Installation Guide*.

  The information center of searchable HTML files is viewable on the web at http://pic.dhe.ibm.com/infocenter/compbg/v121v141/index.jsp.
- PDF documents

  PDF documents are located by default in the /opt/ibmcmp/vacpp/bg/12.1/doc/en_US/pdf/ directory. The PDF files are also available on the web at http://www.ibm.com/software/awdtools/xlcpp/features/bg/library/.

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

Table 3. XL C/C++ PDF files

Document title	PDF file name	Description
IBM XL C/C++ for Blue Gene/Q, V12.1 Installation Guide, GC14-7362-00	install.pdf	Contains information for installing XL C/C++ and configuring your environment for basic compilation and program execution.

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

Document title	PDF file name	Description
Getting Started with IBM XL C/C++ for Blue Gene/Q, V12.1, GC14-7361-00	getstart.pdf	Contains an introduction to the XL C/C++ product, with information on setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
IBM XL C/C++ for Blue Gene/Q, V12.1 Compiler Reference, GC14-7363-00	compiler.pdf	Contains information about the various compiler options, pragmas, macros, environment variables, and built-in functions, including those used for parallel processing.
IBM XL C/C++ for Blue Gene/Q, V12.1 Language Reference, GC14-7364-00	langref.pdf	Contains information about the C and C++ programming languages, as supported by IBM, including language extensions for portability and conformance to nonproprietary standards.
IBM XL C/C++ for Blue Gene/Q, V12.1 Optimization and Programming Guide, SC14-7365-00	proguide.pdf	Contains information on advanced programming topics, such as application porting, interlanguage calls with Fortran code, library development, application optimization and parallelization, and the XL C/C++ high-performance libraries.

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

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

http://www.ibm.com/software/awdtools/xlcpp/features/bg/library/

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

Standards and specifications

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

- *Information Technology - Programming languages - C, ISO/IEC 9899:1990*, also known as *C89*.
- *Information Technology - Programming languages - C, ISO/IEC 9899:1999*, also known as *C99*.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:1998*, also known as *C++98*.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2003(E)*, also known as *Standard C++*.
- *Information Technology - Programming languages - Extensions for the programming language C to support new character data types, ISO/IEC DTR 19769*. This draft technical report has been accepted by the C standards committee, and is available at http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1040.pdf.

- *Draft Technical Report on C++ Library Extensions, ISO/IEC DTR 19768*. This draft technical report has been submitted to the C++ standards committee, and is available at http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2005/n1836.pdf.
- *ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985*.
- *OpenMP Application Program Interface Version 3.1*, available at http://www.openmp.org

Other IBM information

- *Blue Gene/Q Hardware Overview and Installation Planning, SG24-7872*, available at http://www.redbooks.ibm.com/redpieces/abstracts/sg247872.html?Open
- *Blue Gene/Q Hardware Installation and Maintenance Guide, SG24-7974*, available at http://www.redbooks.ibm.com/redpieces/abstracts/sg247974.html?Open
- *Blue Gene/Q High Availability Service Node, REDP-4657*, available at http://www.redbooks.ibm.com/redpieces/abstracts/redp4657.html?Open
- *Blue Gene/Q System Administration, SG24-7869*, available at http://www.redbooks.ibm.com/redpieces/abstracts/sg247869.html?Open
- *Blue Gene/Q Application Development*, *SG24-7948*, available at http://www.redbooks.ibm.com/redpieces/abstracts/sg247948.html?Open
- *Blue Gene/Q Code Development and Tools Interface, REDP-4659*, available at http://www.redbooks.ibm.com/redpieces/abstracts/redp4659.html?Open

Other information

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

Technical support

Additional technical support is available from the XL C/C++ Support page at http://www.ibm.com/software/awdtools/xlcpp/features/bg/support/. This page provides a portal with search capabilities to a large selection of Technotes and other support information.

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

For the latest information about XL C/C++, visit the product information site at http://www.ibm.com/software/awdtools/xlcpp/features/bg/.

How to send your comments

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

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

Chapter 1. Introducing XL C/C++

IBM XL C/C++ for Blue Gene/Q, V12.1 is an advanced, high-performance compiler that can be used for developing complex, computationally intensive programs, including interlanguage calls with C and Fortran programs.

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

About the Blue Gene architecture

The IBM Blue Gene/Q solution is the third generation machine in IBM's Blue Gene® program. It adheres to the key design strategies of the Blue Gene program, providing petaflop scale performance in a package that is efficient in terms of power, cooling and floor space, thereby reducing the total cost of ownership.

Compared to Blue Gene®/L and Blue Gene®/P, Blue Gene/Q extended performance through an increase of processor cores and frequency, and added 4-way SMP functionality, hardware DMA, 10 Gb Ethernet, and aggressive power management. The solution typically combines multiple racks of 1024 compute nodes, each containing an SMP PowerPC® A2 processor. Each processor contains 16 cores for use by the application and one core for use by the system.

Blue Gene/Q provides a standard programming and cross compiling environment, and supports a wide range of IBM and open source software libraries and middleware. The Front End nodes contain the Linux Red Hat Enterprise Linux 6.2 (RHEL 6.2) operating system for developing, compiling, and debugging the high performance applications that run on Blue Gene/Q compute nodes.

For more information about the Blue Gene solution, see "*IBM System Blue Gene Solution: Blue Gene/Q Application Development*" available at http://www.redbooks.ibm.com/redpieces/abstracts/sg247948.html?Open.

Commonality with other IBM compilers

IBM XL C/C++ for Blue Gene/Q, V12.1 is part of a larger family of IBM C, C++, and Fortran compilers.

XL C/C++, together with XL Fortran, comprise the family of XL compilers.

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

Operating system support

IBM XL C/C++ for Blue Gene/Q, V12.1, along with the compiler related tools, supports the Red Hat Enterprise Linux 6.2 (RHEL 6.2) operating system on Blue Gene/Q Front End nodes.

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

The generated object programs and runtime libraries are run on Blue Gene/Q compute nodes with the required software and disk space. Blue Gene/Q compute nodes support the Blue Gene Compute Node Kernel (CNK) operating system.

To exploit the various supported hardware configurations, the compiler provides options to tune the performance of applications specific to the type of hardware that will be used to execute the compiled applications.

A highly configurable compiler

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

Compiler invocation commands

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

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

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

Compiler options

You can choose from a large selection of compiler options to control compiler behavior. Different categories of options help you to debug your applications, optimize and tune application performance, select language levels and extensions for compatibility with non-standard features and behaviors supported by other C or C++ compilers, and perform many other common tasks that would otherwise require changing the source code.

XL C/C++ lets you specify compiler options through a combination of environment variables, compiler configuration files, command line options, and compiler directive statements embedded in your program source.

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

Custom compiler configuration files

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

Your compilation needs may frequently involve specifying compiler option settings other than the default settings provided by XL C/C++. If so, you can use makefiles to define your compiler option settings, or alternatively, you can create custom configuration files to define your own sets of frequently used compiler option settings.

For more information about using custom compiler configuration files, see “Using custom compiler configuration files” on page 29.

Language standard compliance

This section provides language standard compliance information for IBM XL C/C++ for Blue Gene/Q, V12.1.

The compiler supports the following programming language specifications for C/C++:

- ISO/IEC 9899:1999 (C99)
- ISO/IEC 9899:1990 (referred to as C89)
- ISO/IEC 14882:2003 (referred to as Standard C++)
- ISO/IEC 14882:1998, the first official specification of the language (referred to as C++98)

In addition to the standardized language levels, XL C/C++ supports language extensions, including:

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

  See “C++0x features” on page 16 for more details.
- A subset of C1X features

  See “C1X features” on page 22 for more details.

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

Compatibility with GNU

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

For details, see the Compatibility with GNU section in *Getting Started with IBM XL C/C++ for Linux, V12.1*.

Source-code migration and conformance checking

XL C/C++ helps protect your investment in your existing C/C++ source code by providing compiler invocation commands that instruct the compiler to compile your application code to a specific language level.

You can also use the **-qlanglvl** compiler option to specify a given language level, and the compiler will issue warnings, errors, and severe error messages if language or language extension elements in your program source do not conform to that language level.

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

Libraries

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

IBM XL C/C++ for Blue Gene/Q, V12.1 requires the Blue Gene/Q tool chain for cross-compilations. To use the runtime environment that contains the libraries, you must ensure that the Blue Gene/Q tool chain is installed. It is recommended that you install the tool chain to the default directory: `/bgsys/drivers/ppcfloor`

Mathematical Acceleration Subsystem library

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

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

See "Using the Mathematical Acceleration Subsystem" in the *XL C/C++ Optimization and Programming Guide* for more information.

Basic Linear Algebra Subprograms

The Basic Linear Algebra Subprograms (BLAS) set of high-performance algebraic functions are shipped in the libxlopt library. These functions let you:

- Compute the matrix-vector product for a general matrix or its transpose.
- Perform combined matrix multiplication and addition for general matrices or their transposes.

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

Other libraries

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

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

Support for Boost libraries

XL C/C++ for Blue Gene/Q, V12.1 partially supports the Boost V1.47.0 libraries. A patch file is available that modifies the Boost 1.47.0 libraries so that they can be built and used with XL C/C++ applications. The patch or modification file does not extend nor provide additional functionality to the Boost libraries.

To access the patch file for building the Boost libraries, go to this page at http://www.ibm.com/support/docview.wss?uid=swg27006911 and follow the link in the download required Boost modification file section.

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

For more information on support for libraries, search on the XL C/C++ Compiler support page at http://www.ibm.com/software/awdtools/xlcpp/features/bg/support/.

Tools, utilities, and commands

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

Utilities

gxlc and gxlc++ utilities
The **gxlc** and **gxlc++** utilities translate GNU C and GNU C++ invocation commands into corresponding **xlc** and **xlc++** commands before invoking the XL C/C++ compiler. The purpose of these utilities is to minimize the number of changes to makefiles used for existing applications built with the GNU compilers and to facilitate the transition to the XL C/C++ compiler. For more information, see the Tools, utilities, and commands section in *Getting Started with IBM XL C/C++ for Linux, V12.1*.

new_install
The **new_install** utility configures IBM XL C/C++ for Blue Gene/Q, V12.1 for use on your system, after you install the compiler.

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

Commands

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

Program optimization

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

With these options, you can perform the following tasks:
- Select different levels of compiler optimizations.
- Control optimizations for loops, floating point, and other types of operations.
- Optimize a program for a particular class of machines or for a very specific machine configuration, depending on where the program will run.

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

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

For more information, see these related topics:
- "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*
- "Optimizing and tuning options" in the *XL C/C++ Compiler Reference*
- "Compiler built-in functions" in the *XL C/C++ Compiler Reference*

Shared memory parallelization

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

You can use any of the following methods to develop your parallelized applications with XL C/C++:
- Directive-based shared memory parallelization
- Instructing the compiler to automatically generate shared memory parallelization
- Message passing based shared or distributed memory parallelization (MPI)

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

OpenMP directives

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

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

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

As of XL C/C++ for Blue Gene/Q, V12.1, XL C/C++ supports the OpenMP API Version 3.1 specification. See “OpenMP 3.1” on page 23 for an overview of the support provided by this feature.

Speculative execution of threads

Thread-level speculative execution uses hardware support that dynamically detects thread conflicts and rolls back conflicting threads for re-execution. You can get significant performance gains in your applications by adding the compiler directives of thread-level speculative execution without rewriting the program code.

For details, see Thread-level speculative execution.

Transactional memory

Transactional memory is a model for controlling concurrent memory accesses in the scope of parallel programming. It is also called lock-free synchronization and is an alternative to lock-based synchronization. Transactions are implemented through regions of code that you can designate to be single operations for the system.

For details, see Transactional memory.

For more information about program performance optimization, see:
- "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*
- www.openmp.org

Diagnostic listings

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

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

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

Symbolic debugger support

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

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

The debugging information can be examined by **gdb** or any other symbolic debugger that is supported on Blue Gene/Q to help you debug your programs. For how to use **gdb** remotely on the Blue Gene/Q compute nodes, see "*Blue Gene/Q Application Development*" available at http://www.redbooks.ibm.com/redpieces/abstracts/sg247948.html?Open.

Chapter 2. What's new for IBM XL C/C++ for Blue Gene/Q, V12.1

This section describes features and enhancements added to the compiler in IBM XL C/C++ for Blue Gene/Q, V12.1.

The XL compiler for Blue Gene/Q contains some significant enhancements since XL C/C++, V9.0 that supported Blue Gene/L and Blue Gene/P. It also contains many enhancements that are in common with the compilers on the AIX and Linux platforms.

Blue Gene/Q features

This section describes the Blue Gene/Q new features added to the compiler in IBM XL C/C++ for Blue Gene/Q, V12.1.

Quad Processing Extension support

This release of the compiler supports the Quad Processing eXtension (QPX) instruction set of the Blue Gene/Q platform.

With the **-qsimd=auto** option enabled by default, the compiler can automatically take advantage of QPX vector instructions.

New data types and intrinsic functions are introduced to support the QPX instructions. With the QPX intrinsic functions, you can efficiently manipulate vector operations in your application.

The QPX data type is twice larger than the data type of the Double Hummer instruction set present in the Blue Gene/L and Blue Gene/P platforms: four double-precision floating-point values instead of two double-precision floating-point values. It is automatically enabled when you compile your program for the Blue Gene/Q architecture.

You can use the **-qflttrap=qpxstore** option to enable the detection of floating-point exceptions in QPX vectors.

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

Speculative execution of threads

Thread-level speculative execution uses hardware support that dynamically detects thread conflicts and rolls back conflicting threads for re-execution.

You can get significant performance gains in your applications by adding the compiler directives of thread-level speculative execution without rewriting the program code.

Thread-level speculative execution is enabled with the "-qsmp=speculative" compiler option.

Related information

- "-qsmp"
- Thread-level speculative execution
- #pragma speculative for
- #pragma speculative sections
- Built-in functions for thread-level speculative execution
- Environment variables for thread-level speculative execution

Transactional memory

Transactional memory is a model for controlling concurrent memory accesses in the scope of parallel programming. It is also called lock-free synchronization and is an alternative to lock-based synchronization.

In Blue Gene/Q, the transactional memory model is implemented in the hardware to access all the memory up to the 16 GB boundary.

Transactions are implemented through regions of code that you can designate to be single operations for the system.

The transactional memory is enabled with the "-qtm" compiler option.

Related information

- Transactional memory
- #pragma tm_atomic
- Built-in functions for transactional memory
- Environment variables for transactional memory

Compiler options and pragma directives support

This section describes new and changed compiler options and pragma directives for IBM XL C/C++ for Blue Gene/Q, V12.1.

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

Debug optimized program using -g

The **-g** compiler option generates debugging information for use by a symbolic debugger. You can use **-g** to debug optimized code by viewing or possibly modifying accessible variables at selected source locations in the debugger.

The **-g** option has been extended to have different levels (**-g0** - **-g9**) so that you can balance between debug capability and compiler optimization. Higher levels provide a more complete debug support, at the cost of runtime or possible compile-time performance, while lower levels provide higher runtime performance, at the cost of some capability in the debugging session.

You can use **-g** to debug optimized code at **-O2** by viewing or possibly modifying accessible variables at selected source locations in the debugger.

For details, see -g.

Compiler options or pragma directives for Blue Gene/Q

This section summarizes new or changed compiler options and pragma directives that support Blue Gene/Q specific features.

Blue Gene/Q specific compiler options

-qarch
: **-qarch** specifies the processor architecture where the code may run. **-qarch=qp** produces object code that runs on the Blue Gene/Q platform. It is enabled by default.

-qtune
: **-qtune** tunes instruction selection, scheduling, and other architecture-dependent performance enhancements to run best on a specific hardware architecture. **-qtune=qp** specifies that optimizations are tuned for the Blue Gene/Q platform. It is enabled by default.

-qstaticlink
: When **-qstaticlink** is in effect, the compiler links only static libraries with the object file being produced. It is enabled by default. You can specify **-qnostaticlink** to dynamically link your programs.

-qtm
: **-qtm** enables support for transactional memory.

-qsmp
: **-qsmp=speculative** enables support for thread-level speculative execution.

-qnokeyword
: To avoid invading user namespace, you can specify **-qnokeyword=vector4double** to disable the support of `vector4double` data type.

-qflttrap
: **-qflttrap=qpxstore** enables the detection of Not a Number (NaN) or infinity values in Quad Processing eXtension (QPX) vectors. The exceptions only occur on QPX store instructions.

-qsimd
: **-qsimd=auto** enables automatic generation of QPX vector instructions. It is enabled by default at all optimization levels. To disable automatic generation of QPX instructions, use **-qsimd=noauto**.

Blue Gene/Q specific pragma directives

#pragma speculative for
: The **speculative for** pragma instructs the compiler to speculatively parallelize a `for` loop.

#pragma speculative section, #pragma speculative sections
: The **speculative sections** pragma instructs the compiler to speculatively parallelize sections of the code. In code blocks delimited by **speculative sections**, you can use the **speculative section** directive to delimit program code segments.

#pragma tm_atomic
: The **tm_atomic** pragma indicates a transactional atomic region.

#pragma pack
: The **#pragma pack** directive gives members of aggregates an alignment of 1 byte. When **#pragma pack** is applied to an aggregate with a `vector4double` member, the compiler generates a severe error message.

Other new or changed compiler options and pragma directives

This section summarizes other new or changed compiler options and pragma directives since V9.0 of the XL C/C++ compiler.

New or changed compiler options

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

-qfunctrace
: Traces the entry and exit points of functions in a compilation unit or only for a specific list of functions.

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

-qhot

The **-qhot=[no]simd** option has been deprecated and replaced by the **-q[no]simd** option. For details, see -qsimd.

In addition, the **-qhot=fastmath** option has been added to allow your applications to use fast scalar versions of math functions.

-qinclude

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

-qinfo

The suboptions `als` and `noals` have been added to the **qinfo** option to report (or not report) possible violations of the ANSI aliasing rule.**-qinfo=all** now enables all diagnostic messages for all groups except **als** and **ppt**

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

-qinline
: Attempts to inline functions instead of generating calls to those functions, for improved performance.

-qkeyword
: C++0x The new suboption **-q[no]keyword=constexpr** enables or disables the `constexpr` keyword.

-qlanglvl
: The following suboptions are added or updated:

C++0x **-qlanglvl=autotypededuction**
: This suboption controls whether the auto type deduction or the trailing return type feature is enabled. You can use it to delegate the task of type deduction of an auto variable to the compiler from the type of its initializer expression.

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

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

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

C++0x **-qlanglvl=constexpr**
This suboption enables the generalized constant expressions feature, which extends the expressions permitted within constant expressions.

Note: In XL C/C++ V12.1, this feature is a partial implementation of what is defined in the C++0x standard.

C++0x **-qlanglvl=explicitconversionoperators**
This suboption enables the explicit conversion operators feature, which allows you to inhibit unintended implicit conversions through the user-defined conversion function.

C1X **-qlanglvl=extc1x**
This suboption enables all the currently supported C1X features and other implementation-specific language extensions.

C++0x **-qlanglvl=referencecollapsing**
This suboption enables the reference collapsing feature, with which you can form a reference to a reference type using a `decltype` specifier, a `typedef` name, or a template type parameter.

C++0x **-qlanglvl=rightanglebracket**
This suboption enables the right angle bracket feature, which removes the white space requirement for consecutive right angle brackets.

C++0x **-qlanglvl=rvaluereferences**
This suboption enables the rvalue references feature.

C++0x **-qlanglvl=scopedenum**
This suboption enables the scoped enumeration feature, with which you can declare a scoped enumeration type or an enumeration without providing the enumerators.

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

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

C++0x **-qlanglvl=c99longlong**
This suboption controls whether the C99 `long long` feature is enabled. This feature improves source compatibility between the C and C++ languages.

C++0x **-qlanglvl=c99preprocessor**
: This suboption controls whether the C99 preprocessor features adopted in C++0x are enabled. This feature can be used to provide a more common preprocessor interface for C and C++ compilers.

C++0x **-qlanglvl=decltype**
: This suboption controls whether the decltype feature is enabled. This feature can be used to get a type that is based on the resultant type of a possibly type-dependent expression.

C++0x **-qlanglvl=delegatingctors**
: This suboption controls whether the delegating constructors feature is enabled. This feature can be used to concentrate common initializations in one constructor.

C++0x **-qlanglvl=extendedfriend**
: This suboption controls whether the extended friend declarations feature is enabled. This feature can be used to accept additional forms of non-function friend declarations.

C++0x IBM **-qlanglvl=extendedintegersafe**
: This suboption controls whether or not `unsigned long long int` can be used as the type for decimal integer literals that do not have a suffix containing `u` or `U` and cannot be represented by the `long long int` type. This option takes effect only when the **-qlanglvl=c99longlong** option is specified.

C++0x **-qlanglvl=externtemplate**
: This suboption controls whether the explicit instantiation declarations feature is enabled. This feature can be used to suppress the implicit instantiation of a template specialization or its members.

C++0x **-qlanglvl=inlinenamespace**
: This suboption controls whether the inline namespace definitions feature is enabled. This feature can be used to define and specialize members of an inline namespace as if they were also members of the enclosing namespace.

C++0x **-qlanglvl=static_assert**
: This suboption controls whether the static assertions feature is enabled. This feature can be used to produce compile-time assertions for which a severe error message is issued on failure.

C++0x **-qlanglvl=variadic[templates]**
: This suboption controls whether the variadic templates feature is enabled. This feature can be used to define class or function templates that have any number (including zero) of parameters.

See “C++0x features” on page 16 for more information about the new C++0x features.

See “C1X features” on page 22 for more information about the C1X features.

-qlibmpi
: Tunes code based on the known behavior of the Message Passing Interface (MPI) functions.

-qlistfmt
: The **-qlistfmt** option is enhanced to generate HTML reports as well as

XML reports, containing information about optimizations performed by the compiler and missed optimization opportunities.

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

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

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

-qsaveopt
The existing **-qsaveopt** option is enhanced to also include the user's configuration file name and the options specified in the configuration files.

-qreport
When used together with compiler options that enable automatic parallelization or vectorization, the **-qreport** option now reports the number of streams in a loop and produces information when loops cannot be SIMD vectorized due to non-stride-one references.

-qrestrict (C only)
You can use **-qrestrict** to indicate to the compiler that no other pointer can access the same memory that has been addressed by function parameter pointers.

-qshowmacros
When used in conjunction with the **-E** option, the **-qshowmacros** option replaces preprocessed output with macro definitions. There are suboptions provided to control the emissions of predefined and user-defined macros more precisely.

-qsmp=omp
When **-qsmp=omp** is in effect, the additional functionality of OpenMP API 3.1 is now available. For more information, see “OpenMP 3.1” on page 23.

-qstackprotect
Protects your applications against malicious code or programming errors that overwrite or corrupt the stack.

-qstrict

Many suboptions have been added to the **-qstrict** option to allow more control over optimizations and transformations that violate strict program semantics. For details, see “Performance and optimization” on page 24.

-qstrict=vectorprecision disables vectorization in loops where it might produce different results in vectorized iterations than in nonvectorized ones.

-qtimestamps
This option can be used to remove timestamps from generated binaries.

-qtls The thread local storage support has been enhanced to include `__attribute__((tls-model("`*`string`*`")))` where *string* is one of `local-exec`, `initial-exec`, `local-dynamic`, or `global-dynamic`.

New or changed pragma directives

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

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

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

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

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

Language support enhancements

This section describes language enhancements added to the compiler in IBM XL C/C++ for Blue Gene/Q, V12.1.

C++0x features

C++0x is the working draft of the new C++ programming language standard.

Note: C++0x is a new version of the C++ programming language standard. IBM continues to develop and implement the features of the new standard. The implementation of the language level is based on IBM's interpretation of the standard. Until IBM's implementation of all the features of the C++0x standard is complete, including the support of a new C++ standard library, the implementation may change from release to release. IBM makes no attempt to maintain compatibility, in source, binary, or listings and other compiler interfaces, with earlier releases of IBM's implementation of the new features of the C++0x standard and therefore they should not be relied on as a stable programming interface.

Note: C++0x has been ratified and published as ISO/IEC 14882:2011. All references to C++0x in this document are equivalent to the ISO/IEC 14882:2011 standard. Corresponding information, including programming interfaces, will be updated in a future release.

The following features are introduced:

- A new language level: `extended0x`
- C99 `long long` and new integer promotion rules for arithmetic conversions
- C++ preprocessor support for C99 features
- C99 preprocessor features adopted in C++0x
- Auto type deduction
- Decltype
- Delegating constructors
- Explicit instantiation declarations
- Extended friend declarations

- Inline namespace definitions
- Static assertion
- Variadic templates
- Explicit conversion operators
- Generalized constant expressions
- Reference collapsing
- Right angle brackets
- Rvalue references
- Scoped enumerations
- Trailing return type

New language level - extended0x

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

A new suboption has been added to the **-qlanglvl** option. You can use the **-qlanglvl=extended0x** option to enable most of the C++ features and all the currently-supported C++0x features. For details, see -qlanglvl in the *XL C/C++ Compiler Reference*.

C99 long long under C++

The C99 `long long` feature improves source compatibility between the C and C++ languages. To enable C99 `long long`, use the **-qlanglvl=c99longlong** option under the XL C++ compiler.

IBM When C99 `long long` is in effect, if a decimal integer literal that does not have a suffix containing `u` or `U` cannot be represented by the `long long int` type, you can specify the `-qlanglvl=[no]extendedintegersafe` option to determine whether to use the `unsigned long long int` type to represent the literal. For more information, see "Integer literals" in the *XL C/C++ Language Reference*.

In this release, compiler behavior changes when performing certain arithmetic operations with integral literal data types. Specifically, the integer promotion rules have changed.

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

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

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

```
int
long int
long long int
```

```
unsigned long long
```

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

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

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

Preprocessor changes

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

- Regular string literals can now be concatenated with wide-string literals.
- The `#line <integer>` preprocessor directive has a larger upper limit. It has been increased from 32767 to 2147483647 for C++ .
- C++ now supports `_Pragma` operator.
- These macros now apply to C++ as well as C:
  - __C99_MACRO_WITH_VA_ARGS (also available with **-qlanglvl=extended**)
  - __C99_MAX_LINE_NUMBER (also available with **-qlanglvl=extended**)
  - __C99_PRAGMA_OPERATOR
  - __C99_MIXED_STRING_CONCAT

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

C99 preprocessor features adopted in C++0x

With several C99 preprocessor features adopted in C++0x, C and C++ compilers provide a more common preprocessor interface, which can ease porting C source files to the C++ compiler, eliminate semantic differences between the C and C++ preprocessors, and avoid preprocessor compatibility issues or diverging preprocessor behaviors.

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

For more information, see "C99 preprocessor features adopted in C++0x)" in the *XL C/C++ Language Reference*.

Auto type deduction

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

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

For more information, see "The auto type specifier (C++0x)" in the *XL C/C++ Language Reference*.

Decltype

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

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

For more information, see "The decltype(expression) type specifier (C++0x)" in the *XL C/C++ Language Reference*.

Delegating constructors

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

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

For more information, see "Delegating constructors (C++0x)" in the *XL C/C++ Language Reference*.

Explicit instantiation declarations

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

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

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

Extended friend declarations

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

- Template parameters, `typedef` names, and basic types can be declared as friends.
- The class-key in the context for friend declarations is no longer necessary in C++0x.

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

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

Inline namespace definitions

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

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

For more information, see "Inline namespace definitions (C++0x)" in the *XL C/C++ Language Reference*.

Static assertion

The static assertion feature provides you with the following benefits:

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

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

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

For more information, see "static_assert declaration (C++0x)" in the *XL C/C++ Language Reference*.

Variadic templates

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

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

For more information, see "Variadic templates (C++0x)" in the *XL C/C++ Language Reference*.

Explicit conversion operators

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

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

For more information, see "Explicit Conversion Operators (C++0x)" in the *XL C/C++ Language Reference*.

Generalized constant expressions

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

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

Note: In XL C/C++ V12.1, this feature is a partial implementation of what is defined in the C++0x standard.

Reference collapsing

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

- A `decltype` specifier
- A `typedef` name

- A template type parameter

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

For more information, see "Reference collapsing (C++0x)" in the *XL C/C++ Language Reference*.

Right angle brackets

In the C++ language, two consecutive closing angle brackets (>) must be separated with a white space, because they are otherwise parsed as the bitwise right-shift operator (>>). The right angle bracket feature removes the white space requirement for consecutive right angle brackets, thus making programming more convenient.

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

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

Rvalue references

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

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

For more information, see "Using rvalue references (C++0x)" in the *XL C/C++ Optimization and Programming Guide*.

Scoped enumerations

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

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

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

Trailing return type

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

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

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

For more information, see "Trailing return type (C++0x)" in the *XL C/C++ Language Reference*.

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

-qlanglvl

C1X features

This release introduces support for selected features of C1X.

Note: C1X is a new version of the C programming language standard. IBM continues to develop and implement the features of the new standard. The implementation of the language level is based on IBM's interpretation of the standard. Until IBM's implementation of all the features of the C1X standard is complete, including the support of a new C standard library, the implementation may change from release to release. IBM makes no attempt to maintain compatibility, in source, binary, or listings and other compiler interfaces, with earlier releases of IBM's implementation of the new features of the C1X standard and therefore they should not be relied on as a stable programming interface.

Note: C1X has been ratified and published as ISO/IEC 9899:2011. All references to C1X in this document are equivalent to the ISO/IEC 9899:2011 standard. Corresponding information, including programming interfaces, will be updated in a future release.

The following features are introduced in IBM XL C/C++ for Blue Gene/Q, V12.1:
- Anonymous structures
- Complex type initialization
- New language level - **extc1x**
- The `_Noreturn` function specifier
- Static assertions

Anonymous structures

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

Complex type initialization

Macros CMPLX, CMPLXF, and CMPLXL are defined inside the standard header file complex.h to enable the initialization of complex types under the extc1x language level. For more information, see "Initialization of complex types (C1X)" in the *XL C/C++ Language Reference*.

New language level - extc1x

A new suboption has been added to the **-qlanglvl** option in this release. When you compile with the C compiler, you can use **-qlanglvl=extc1x** to enable C1X features that are currently supported by XL C/C++. Certain C1X features are also available when you compile with the C++ compiler. For further information, see the sections that describe individual features.

The `_Noreturn` function specifier

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

Static assertions

The addition of static assertions to the C language has the following benefits:
- Libraries can detect common usage errors at compile time.
- Implementations of the C Standard Library can detect and diagnose common usage errors, improving usability.

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

For more information, see "_Static_assert declaration (C1X)" in the *XL C/C++ Language Reference*.

OpenMP 3.1

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

OpenMP 3.1 includes the following updates to OpenMP 3.0:
- Adds `final` and `mergeable` clauses to the `task` construct to support optimization.
- Adds the `taskyield` construct to allow users to specify where in the program can perform task switching.
- Adds the `omp_in_final` runtime library function to support specialization of final task regions.
- Extends the `atomic` construct to include `read`, `write`, and `capture` forms; adds the `update` clause to apply the existing form of the `atomic` construct.
- Adds two reduction operators: `min` and `max`.
- Allows const-qualified types to be specified on the `firstprivate` clause.
- Adds the `OMP_PROC_BIND` environment variable to control whether OpenMP threads are allowed to move between processors.
- Extends the `OMP_NUM_THREADS` environment variable to specify the number of threads to use for nested parallel regions.

Related information
- "OpenMP environment variables" in the *XL C/C++ Compiler Reference*
- "Pragma directives for parallel processing" in the *XL C/C++ Compiler Reference*

- www.openmp.org

New built-in functions

This section describes the major categories of built-in functions that are newly added since XL C/C++, V9.0.

Thread-level speculative execution built-in functions

With the built-in functions provided for thread-level speculative execution, you can retrieve statistical information, write statistics into log files, or switch mode to transactional memory (TM) at runtime.

Transactional memory built-in functions

With the built-in functions provided for transactional memory, you can retrieve statistical information, write statistics into log files, or switch mode to thread-level speculative execution (SE) at runtime.

Quad Processing eXtension built-in functions

You can use the Quad Processing Extension (QPX) built-in functions to access individual elements of vectors and manipulate vectors.

GCC atomic memory access built-in functions

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

Conversion functions

These new functions convert between Declets and Binary Coded Decimal.

- `__cbcdtd`
- `__cdtbcd`

Comparison functions

This new function compares bytes.

- `__cmpb`

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

Performance and optimization

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

Enhancements to -qstrict

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

You can use 16 new suboptions separately or use a suboption group. For detailed information about these suboptions, see "-qstrict" in the *XL C/C++ Compiler Reference*.

Reports about compiler optimizations

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

Performance-related compiler options and directives

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

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

Table 4. Performance-related compiler options and directives

-qinline=level=*number*	A new option is added to -qinline to provide guidance to the compiler about the relative value of inlining in relation to the default value of 5.*number* is a range of integer values 0 - 10 that indicates the level of inlining you want to use. For details, see -qinline in the *XL C/C++ Compiler Reference*.
-qfloat	Some **-qfloat** suboptions are affected by the new suboptions for **-qstrict**.

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

New diagnostic reports

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

Compiler reports in XML or HTML format

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

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

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

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

Report of data reorganization

The compiler can generate the following information in the listing files:
- Data reorganizations (a summary of how program variable data gets reorganized by the compiler)
- The location of data prefetch instructions inserted by the compiler

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

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

Additional loop analysis

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

New and enhanced diagnostic options

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

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

Table 5. Listings-related compiler options and directives

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

Chapter 3. Setting up and customizing XL C/C++

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

Using custom compiler configuration files

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

You have the following options to customize compiler settings:

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

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

Chapter 4. Developing applications with XL C/C++

C/C++ application development consists of repeating cycles of editing, compiling and linking (by default a single step combined with compiling), and running.

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

The compiler phases

A typical compiler invocation executes some or all of these activities in sequence. For link time optimizations, some activities will be executed more than once during a compilation. As each program runs, the results are sent to the next step in the sequence.
1. Preprocessing of source files
2. Compilation, which may consist of the following phases, depending on what compiler options are specified:
   a. Front-end parsing and semantic analysis
   b. High-level optimization
   c. Low-level optimization
   d. Register allocation
   e. Final assembly
3. Assemble the assembly (**.s**) files, and the unpreprocessed assembler (**.S**) files after they are preprocessed
4. Object linking to create an executable application

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

Editing C/C++ source files

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

Source programs must be saved using a recognized file name suffix. See "XL C/C++ input and output files" on page 34 for a list of suffixes recognized by XL C/C++.

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

Compiling with XL C/C++

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

Invoking the compiler

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

To compile a source program, use any of the available XL C/C++ for Blue Gene/Q compiler invocation commands. The **bg**-prefixed invocation commands on the Blue Gene/Q Front End node (RHEL 6.2) are for cross-compiling applications for use on the Blue Gene/Q compute node. The invocation commands that are not prefixed with **bg** create executable programs targeted for RHEL 6.2 on POWER® platforms, and are provided only for testing and debugging purposes. For the development of applications targeted for the Front End node, IBM provides the IBM XL C/C++ for Linux, V12.1 product. As well, only the compiler options which are supported by the **bg** cross-compiler commands are supported when using these compiler invocations to create executable files for Blue Gene/Q.

To specify an invocation command, use the following basic syntax:

```
                                   ┌───────────────────────────────────────────┐
                                   │ ┌───────────────────────┐                 │
            (1)                    ▼ ▼                       │                 │
►►──┬──────────bgxlc─────┬───────────┬─────────────────┬─────┴──input_file─────┴────────►◄
    │  (2)               │           └─compiler_option─┘
    └──────────┬─bgxlC───┘
               └─bgxlc++─┘
```

Notes:

1 Basic invocation to compile C source code

2 Basic invocations to compile C++ source code

For most applications, you should compile with **bgxlc**, **bgxlc++**, or a thread safe counterpart. You can use **bgxlc++** to compile either C or C++ program source, but compiling C++ files with **bgxlc** may result in link or run time errors because libraries required for C++ code are not specified when the linker is called by the C compiler.

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

Compiling parallelized XL C/C++ applications

XL C/C++ provides thread-safe compiler invocation commands that you can use when compiling parallelized applications for use in multiprocessor environments.

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

- bgxlC_r
- bgxlc++_r
- bgxlc_r

XL C/C++ provides additional thread-safe invocations to meet specific compilation requirements. See "Invoking the compiler" in the *XL C/C++ Compiler Reference* for more information.

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

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

Specifying compiler options

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

You can specify compiler options:

- On the command-line with command-line compiler options
- In your source code using directive statements
- In a makefile
- In the stanzas found in a compiler configuration file
- Or by using any combination of these techniques

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

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

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

Note: Some compiler options do not follow the priority sequence described above.

For example, the **-I** compiler option is a special case. The compiler searches any directories specified with **-I** in the vac.cfg file before it searches the directories specified with **-I** on the command-line. The option is cumulative rather than preemptive.

See the *XL C/C++ Compiler Reference* for more information about compiler options and their usage.

Other options with cumulative behavior are **-R** and **-l** (lowercase L).

You can also pass compiler options to the linker, assembler, and preprocessor. See "Compiler options reference" in the *XL C/C++ Compiler Reference* for more information about compiler options and how to specify them.

XL C/C++ input and output files

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

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

Table 6. Input file types

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

Table 7. Output file types

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

Linking your compiled applications with XL C/C++

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

For example, running the following command:

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

compiles `file1.C` and `file3.C` to produce the object files `file1.o` and `file3.o`, then all object files (including `file2.o`) are submitted to the linker to produce one executable.

Compiling and linking in separate steps

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

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

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

- "Linking" in the *XL C/C++ Compiler Reference*

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

Dynamic and static linking

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

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

Static linking means that the code for all routines called by your program becomes part of the executable file. On Blue Gene/Q platforms, static linking is enabled by default.

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

Related information

- The -qstaticlink compiler option

Running your compiled application

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

To run a program on Blue Gene/Q, use `runjob` and enter the name of the program executable file with any runtime arguments on the command line. For details about the `runjob` command, see "*Blue Gene/Q Application Development*" available at http://www.redbooks.ibm.com/redpieces/abstracts/sg247948.html?Open.

Canceling execution

To cancel a running program on Blue Gene/Q, you can cancel the `runjob` command. For details, see "*Blue Gene/Q Application Development*" available at http://www.redbooks.ibm.com/redpieces/abstracts/sg247948.html?Open.

You can press the **Ctrl+C** key to stop the job that is specified by `runjob`.

Setting runtime options

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

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

XL C/C++ compiler diagnostic aids

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

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

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

Debugging compiled applications

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

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

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

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

Determining what level of XL C/C++ is installed

When contacting software support for assistance, you will need to know what level of XL C/C++ is installed on your machine.

To display the version and release level of the compiler you have installed on your system, invoke the compiler with the **-qversion** compiler option.

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

```
bgxlc++ -qversion=verbose
```

Notices

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

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

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

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

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

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

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

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

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

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, 2010.

Trademarks and service marks

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

Adobe, the Adobe logo, PostScript, and the PostScript 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.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

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

Other company, product, or service names may be trademarks or service marks of others.

Index

Product Number: 5799-AG1

Printed in USA

GC14-7361-00