IBM XL C/C++ Advanced Edition for Linux, V9.0

Getting Started with XL C/C++

GC23-5891-00

IBM XL C/C++ Advanced Edition for Linux, V9.0

Getting Started with XL C/C++

GC23-5891-00

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 23.

First Edition

This edition applies to IBM XL C/C++ Advanced Edition for Linux, V9.0 (Program number 5724-S73) 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 v
Who should read this document v
How to use this document. v
Conventions used in this document v
Related information viii
IBM XL C/C++ publications viii
Other IBM publications ix
Other publications ix
Technical support ix
How to send your comments x

Chapter 1. Introducing XL C/C++ 1
Commonality with other IBM compilers 1
Hardware and operating system support 1
A highly configurable compiler 1
Language standards compliance 2
Compatibility with GNU 2
Source-code migration and conformance checking 3
Libraries 3
Mathematical Acceleration Subsystem library . . 4
Basic Linear Algebra Subprograms 4
Tools and utilities. 4
Program optimization 5
64-bit object capability 5
Shared memory parallelization 6
OpenMP directives 6
Diagnostic listings 6
Symbolic debugger support 7

Chapter 2. What's new for IBM XL C/C++ Advanced Edition for Linux, V9.0 9
C/C++ language-related updates 9
Default language level changed for C - extc99 . . 9
Arithmetic conversions with long long data types 9
Architecture and processor support 10
New default setting for -qtune 10
New support for POWER6 processors 10
Performance and optimization 10
Performance-related compiler options and directives 10
Built-in functions new for this release 12
Other new or changed compiler options 12

Chapter 3. Setting up and customizing XL C/C++. 15
Using custom compiler configuration files 15
Determining what level of XL C/C++ is installed. . 15

Chapter 4. Developing applications with XL C/C++ 17
The compiler phases 17
Editing C/C++ source files 17
Compiling with XL C/C++ 18
Invoking the compiler 18
Compiling parallelized XL C/C++ applications 18
Specifying compiler options 19
XL C/C++ input and output files 20
Linking your compiled applications with XL C/C++ 20
Compiling and linking in separate steps 21
Dynamic and static linking 21
Running your compiled application 21
Canceling execution 21
Setting runtime options 22
Running compiled applications on other systems 22
XL C/C++ compiler diagnostic aids 22
Debugging compiled applications 22

Notices 23
Trademarks and service marks 25
Industry standards 25

Index 27

About this document

This document contains overview and basic usage information for the IBM® XL C/C++ Advanced Edition for Linux®, V9.0 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 language, 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 the XL C/C++ compiler.

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 used in this document."

Throughout this document, the **xlc** and **xlc++** 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++ Programming Guide* for detailed information on developing applications with XL C/C++, with a focus on program portability and optimization.

Conventions used in this document

Typographical conventions

The following table explains the typographical conventions used in this document.

Table 1. Typographical conventions

Typeface	Indicates	Example
bold	Lowercase commands, executable names, compiler options and directives.	If you specify **-O3**, the compiler assumes **-qhot=level=0**. To prevent all HOT optimizations with **-O3**, you must specify **-qnohot**.
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.
`monospace`	Programming keywords and library functions, compiler built-in functions, examples of program code, command strings, or user-defined names.	If one or two cases of a `switch` statement are typically executed much more frequently than other cases, break out those cases by handling them separately before the `switch` statement.

Icons

All features described in this document apply to both C and C++ languages. Where a feature is exclusive to one language, or where functionality differs between languages, the following icons are used:

The text describes a feature that is supported in the C language only; or describes behavior that is specific to the C language.

The text describes a feature that is supported in the C++ language only; or describes behavior that is specific to the C++ language.

Syntax diagrams

Throughout this document, 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.

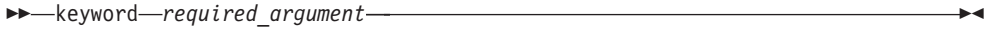
  The ▸▸── symbol indicates the beginning of a command, directive, or statement.

  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):

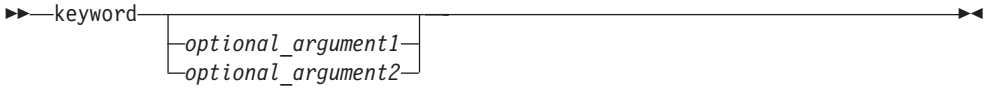


- Optional items are shown below the main path:

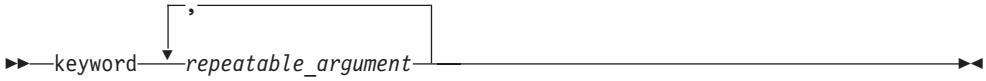
- If you can choose from two or more items, they are shown vertically, in a stack.

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

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



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

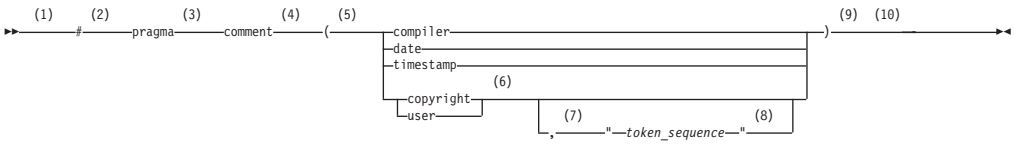


- The item that is the default is shown above the main path.

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

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

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

Related information

The following sections provide information on documentation related to XL C/C++:

- "IBM XL C/C++ publications"
- "Other IBM publications" on page ix
- "Other publications" on page ix

IBM XL C/C++ publications

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

- README files

  README files contain late-breaking information, including changes and corrections to the product documentation. 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 *XL C/C++ 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 *XL C/C++ Installation Guide*. The information center is also viewable on the Web at http://publib.boulder.ibm.com/infocenter/lnxphelp/v9v111/index.jsp.

- PDF documents

PDF documents are located by default in the /opt/ibmcmp/vac/9.0/doc/*LANG*/pdf/ directory, where *LANG* is one of en_US, zh_CN, or ja_JP. The PDF files are also available on the Web at http://www.ibm.com/software/awdtools/xlcpp/library.

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

Table 2. XL C/C++ PDF files

Document title	PDF file name	Description
IBM XL C/C++ Advanced Edition for Linux, V9.0 Installation Guide, GC23-5893-00	install.pdf	Contains information for installing XL C/C++ and configuring your environment for basic compilation and program execution.
Getting Started with IBM XL C/C++ Advanced Edition for Linux, V9.0, GC23-5891-00	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++ Advanced Edition for Linux, V9.0 Compiler Reference, SC23-5889-00	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++ Advanced Edition for Linux, V9.0 Language Reference, SC23-5892-00	langref.pdf	Contains information about the C and C++ programming languages, as supported by IBM, including language extensions for portability and conformance to non-proprietary standards.
IBM XL C/C++ Advanced Edition for Linux, V9.0 Programming Guide, SC23-5890-00	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 Web site at http://www.adobe.com.

More documentation related to XL C/C++ including redbooks, white papers, tutorials, and other articles, is available on the Web at:

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

Other IBM publications

- *ESSL for Linux on POWER V4.2 Guide and Reference*, SA22-7904, available at http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp

Other publications

- *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/support. This page provides a portal with search capabilities to a large selection of technical support FAQs and other support documents.

If you cannot find what you need, you can send e-mail 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.

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 document or any other XL C/C++ documentation, send your comments by e-mail to compinfo@ca.ibm.com.

Be sure to include the name of the document, the part number of the document, 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++ Advanced Edition for Linux, V9.0 is an advanced, high-performance compiler that can be used for developing complex, computationally intensive programs, including interlanguage calls with 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.

Commonality with other IBM compilers

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

IBM XL C/C++ Advanced Edition for Linux, V9.0 is part of a larger family of IBM C, C++, and Fortran compilers.

These compilers are derived from a common code base that shares compiler function and optimization technologies on a variety of platforms and programming languages, such as AIX®, i5/OS®, selected Linux distributions, z/OS®, and z/VM® operating systems. 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.

Hardware and operating system support

V9.0 of XL C/C++ supports several Linux distributions. See the README file and "Before installing XL C/C++" in the *XL C/C++ Installation Guide* for a complete list of requirements.

The compiler, its libraries, and its generated object programs will run on POWER3™, POWER4™, POWER5™, POWER5+™, POWER6™, PowerPC®, and PowerPC 970 systems with the required software and disk space.

To take maximum advantage of the various supported hardware configurations, the compiler provides options to performance-tune applications specifically to the type of hardware that will be used to execute your compiled applications.

A highly configurable compiler

XL C/C++ offers you a wealth of features to let you tailor the compiler to your own unique compilation requirements.

Compiler invocation commands

XL C/C++ provides several different commands that you can use to invoke the compiler, for example, **xlC**, **xlc++**, and **xlc**. 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, **xlc_r** and **xlC_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 call for 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.

See "Using custom compiler configuration files" on page 15 for more information.

Language standards compliance

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 V2.5 extensions to support portable parallelized programming
- Language extensions to support vector programming
- A subset of GNU C and C++ language 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++**.

This support is available when the **gxlc** or **gxlc++** invocation command is used together with select GNU compiler options. Where possible, the compiler maps GNU options to their XL C/C++ compiler option counterparts before invoking the compiler.

These invocation commands use a plain text configuration file to control GNU-to-XL C/C++ option mappings and defaults. You can customize this configuration file to better meet the needs of any unique compilation requirements you may have. See "Reusing GNU C/C++ compiler options with gxlc and gxlc++" in the *XL C/C++ Compiler Reference* for more information.

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

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

Some additional noteworthy points about this relationship are:

- IBM built-in functions coexist with GNU C built-ins.
- Compilation of C and C++ programs uses the GNU C and GNU C++ header files.
- Compilation uses the GNU assembler for assembler input files.
- Compiled C code is linked to the GNU C runtime libraries.
- Compiled C++ code is linked to the GNU C and GNU C++ runtime libraries.
- Debugging uses the GNU debugger, **gdb**

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 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++ ships with the following libraries:

- SMP Runtime Library supports both explicit and automated parallel processing.
- Mathematical Acceleration Subsystem (MASS) library of tuned mathematical intrinsic functions, for 32-bit and 64-bit modes.
- Basic Linear Algebra Subprograms (BLAS) library of tuned algebraic functions.
- C++ C++ Runtime Library contains support routines needed by the compiler.

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 both 32-bit and 64-bit compilation modes, 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++ 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++ Programming Guide*.

Tools and utilities

new_install
: After you install IBM XL C/C++ Advanced Edition for Linux, V9.0, running this utility will configure the compiler for use on your system.

vac_configure
: Use this utility to create additional compiler configuration files that you can then modify to contain your own custom sets of compiler option default settings.

cleanpdf command
: A command related to profile-directed feedback (PDF), **cleanpdf** removes all profiling information from the directory to which profile-directed feedback data is written.

mergepdf command
: A command related to profile-directed feedback (PDF), **mergepdf** provides the ability to weight the importance of two or more PDF records when combining them into a single record. The PDF records must be derived from the same executable.

resetpdf command
: The current behavior of the cleanpdf command is the same as the **resetpdf** comman,d and is retained for compatibility with earlier releases on other platforms.

showpdf command
: The **showpdf** command displays the call and block counts for all procedures executed in a profile-directed feedback training run (compilation under the options **-qpdf1** and **-qshowpdf**).

gxlc and gxlc++ utilities

The **gxlc** and **gxlc++** invocations translate GNU C or GNU C++ invocation commands into corresponding **xlc** or **xlc++** commands before invoking the IBM XL C/C++ Advanced Edition for Linux, V9.0 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 IBM XL C/C++ Advanced Edition for Linux, V9.0.

Program optimization

XL C/C++ provides several compiler options that can help you control the optimization of your programs. With these options, you can:

- 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. C/C++ provides a portfolio of optimizing transformations tailored to various supported hardware. These transformations can:

- Reduce the number of instructions executed for critical operations.
- Restructure generated object code to make optimal use of the PowerPC architecture.
- Improve the usage of the memory subsystem.
- Exploit the ability of the architecture to handle large amounts of shared memory parallelization.

Significant performance improvements are possible with relatively little development effort because the compiler is capable of sophisticated program analysis and transformation. Moreover, XL C/C++ enables programming models, such as OpenMP, which allow you to write high-performance parallel code.

For more information, see:

- "Optimizing your applications" in the *XL C/C++ Programming Guide*
- "Optimization and tuning options" in the *XL C/C++ Compiler Reference*
- "Compiler built-in functions" in the *XL C/C++ Compiler Reference*

64-bit object capability

The XL C/C++ compiler's 64-bit object capability addresses increasing demand for larger storage requirements and greater processing power. The Linux operating system provides an environment that allows you to develop and execute programs that exploit 64-bit processors through the use of 64-bit address spaces.

To support larger executables that can be fit within a 64-bit address space, a separate 64-bit object form is used. The linker binds these objects to create 64-bit executables. Objects that are bound together must all be of the same object format. The following scenarios are not permitted and will fail to load, or execute, or both:

- A 64-bit object or executable that has references to symbols from a 32-bit library or shared library
- A 32-bit object or executable that has references to symbols from a 64-bit library or shared library
- A 64-bit executable that explicitly attempts to load a 32-bit module

- A 32-bit executable that explicitly attempts to load a 64-bit module

XL C/C++ supports 64-bit mode mainly through the use of the **-q64** and **-qarch** compiler options. This combination determines the bit mode and instruction set for the target architecture.

For more information, see "Using 32-bit and 64-bit modes" in the *XL C/C++ Programming Guide*.

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++ 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 requires 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 SMP processors.
3. Directives are available to control synchronization between the processors.

XL C/C++ supports the OpenMP API Version 2.5 specification.

For more information, see:

- "Optimizing your applications" in the *XL C/C++ Programming Guide*
- www.openmp.org

Diagnostic listings

The compiler output listing can 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, refer to ″Compiler messages and listings″ in the *XL C/C++ Compiler Reference*.

Symbolic debugger support

You can instruct XL C/C++ to include debugging information in your compiled objects. That information can be examined by **gdb** or any other symbolic debugger to help you debug your programs.

Chapter 2. What's new for IBM XL C/C++ Advanced Edition for Linux, V9.0

This section describes new features and enhancements in IBM XL C/C++ Advanced Edition for Linux, V9.0.

C/C++ language-related updates

This release changes the default language level for C compilations, and introduces new behavior when doing arithmetic conversions with `long long` data types.

Default language level changed for C - extc99

The default **-qlanglvl** compiler option setting is now **extc99** when invoking the C compiler with the **xlc** invocation. This change will allow you to use C99 features and headers without having to explicitly specify the **extc99** suboption.

You might encounter issues with the following when compiling with the new default **-qlanglvl=extc99** setting:

- Pointers can be qualified with **restrict** in C99, so **restrict** can not be used as an identifier.
- C99 treatment of **long long** data differs from the way **long long** data is handled in C89.
- C99 header files define new macros: LLONG_MAX in limits.h, and va_copy in stdarg.h.
- The value of macro __STD_VERSION__ changes from 199409 to 19990.

To revert to previous **xlc** behavior, specify **-qlanglvl=extc89** when invoking the compiler.

Arithmetic conversions with long long data types

With this release of XL C/C++ V9.0, compiler behavior changes when performing certain arithmetic operations with `long long` data types.

Assume an arithmetic expression where:

- One operand has type `long long int` or `long long`, and,
- The other operand has type `unsigned long int`, but its value cannot be represented in a `long long int` or `long long`.

Previous releases of XL C/C++ will convert both operands to type `long long`.

Starting with this release, the compiler will now convert both operands into type `unsigned long long int` or `unsigned long long`. This new behavior is consistent with GCC compiler behavior.

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

Architecture and processor support

The **-qarch** and **-qtune** compiler options control the code generated by the compiler. These compiler options adjust the instructions, scheduling, and other optimizations to give the best performance for a specified target processor or range of processors.

New default setting for -qtune

The new default **-qtune** settings is:

- -qtune=balanced

The **-qtune=balanced** suboption is new for this release, and becomes the default **-qtune** setting when certain **-qarch** settings are specified. Using **-qtune=balanced** instructs the compiler to tune generated code for optimal performance across a range of recent processor architectures, including POWER6.

New support for POWER6 processors

XL C/C++ 9.0 expands the list of **-qarch** and **-qtune** suboptions to support the newly-available POWER6 processors.

The following **-qarch** and **-qtune** options are now available:

- -qarch=pwr6
- -qarch=pwr6e
- -qtune=pwr6

The **-qipa** compiler option also adds a new architecture cloning suboption to support interprocedural analysis (IPA) optimizations on POWER6 processors:

- -qipa=clonearch=pwr6

Performance and optimization

Many new features and enhancements fall into the category of performance and optimization tuning.

Performance-related compiler options and directives

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

Information presented here is just a brief overview. For more information about these and other performance-related compiler options, refer to Optimization and tuning options in the *XL C/C++ Compiler Reference*.

Table 3. Performance-related compiler options and directives

Option/directive	Description
-qalias= **global** | **noglobal**	These new **-qalias** suboptions enable or disable the application of language-specific aliasing rules across compilation units during link time optimization.

Table 3. Performance-related compiler options and directives (continued)

Option/directive	Description
-qalias= restrict | norestrict	These new **-qalias** suboptions enable or disable optimization for restrict qualified pointers. Specifying **-qalias=restrict** will usually improve performance for code that uses restrict qualified pointers. You can use **-qalias=norestrict** to preserve compatibility with code compiled with versions of the compiler previous to V9.0.
-qnofdpr | -qfdpr	Specifying the **-qfdpr** option instructs the compiler to store optimization information in the created object file. This information is used by the Feedback Directed Program Restructuring (FDPR) performance-tuning utility.
-qfloat= fenv | nofenv	These new **-qfloat** suboptions inform the compiler if code has a dependency on the floating-point hardware environment, such as explicitly reading or writing the floating-point status and control register. Specifying **-qfloat=nofenv** indicates that there is no dependency on the hardware environment, allowing the compiler to perform aggressive optimizations.
-qfloat= gcclongdouble | nogcclongdouble	These new **-qfloat** suboptions have effect only when the **-qldbl128** option is in effect. They instruct the compiler to use either GCC-supplied or IBM-supplied library functions for 128-bit long double operations.
-qfloat= hscmplx | nohscmplx	Specifying **-qfloat=hscmplx** improves optimization of operations involving complex division and complex absolute values.
-qfloat= rngchk | norngchk	Specifying **-qfloat=rngchk** enables range checking on input arguments for software divide and inlined sqrt operations. Specifying **-qfloat=norngchk** instructs the compiler to skip range checking, allowing for better performance in certain circumstances. Specifying the **-qnostrict** compiler option sets **-qfloat=norngchk**.
-qipa=clonearch=pwr6	The **-qipa=clonearch** compiler option now includes a new **pwr6** suboption to support interprocedural analysis (IPA) optimizations on POWER6 processors.
-qipa=threads= [auto | noauto | *number*]	This new **-qipa** suboption lets you specify how many threads the compiler will assign to code generation during the second IPA pass.
-qnoldbl128 | -qldbl128	Specifying **-qldbl128** increases the size of long double types from 64 bits to 128 bits.
-qpdf	The **-qpdf** option can now be used to provide profile-directed feedback on specific objects. See "Object level profile-directed feedback" in the *XL C/C++ Programming Guide* for more information.
-qsmp= threshold=*n*	When **-qsmp=auto** is in effect, this new suboption lets you specify the amount of work required in a loop before the compiler will consider it for automatic parallelization.
#pragma expected_value(*param*, *value***)**	Use the **#pragma expected_value** directive to specify a value that a parameter passed in a function call is most likely to take at run time. The compiler can use this information to perform certain optimizations, such as function cloning and inlining.

Built-in functions new for this release

This section lists built-in functions that are new for this release. For more information on built-in functions provided by XL C/C++, see "Compiler built-in functions" in the *XL C/C++ Compiler Reference*.

Conversion functions

These new functions convert long double data types from IBM style to GCC style.

- `long double __ibm2gccldbl (long double);`
- `_Complex long double __ibm2gccldbl_cmplx (_Complex long double);`

PowerPC cache control

The PowerPC architecture specifies the **dcbst** and **dcbf** cache copy instructions. The following new built-in functions provide direct programmer access to these instructions.

- `void __dcbst(const void* addr);  /* Data Cache Block Store */`
- `void __dcbf(const void* addr);  /* Data Cache Block Flush */`

POWER6 prefetch extensions and cache control

The POWER6 processor has cache control and stream prefetch extensions with support for store stream prefetch and prefetch depth control. XL C/C++ provides the following new built-in functions to provide direct programmer access to these instructions.

- `void __dcbfl(const void* addr);  /* pwr6 - Data Cache Block Flush from L1 data cache only */`
- `void __protected_unlimited_stream_set(unsigned int direction, const void* addr, unsigned int ID);  /* Supported by pwr5 and pwr6 */`
- `void __protected_unlimited_store_stream_set(unsigned int direction, const void* addr, unsigned int ID);  /* Supported by pwr6 */`
- `void __protected_store_stream_set(unsigned int direction, const void* addr, unsigned int ID);  /* Supported by pwr6 */`
- `void __protected_stream_count_depth(unsigned int unit_cnt, unsigned int prefetch_depth, unsigned int ID);  /* Supported by pwr6 */`

Other new or changed compiler options

Compiler options can be specified on the command line or through 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.

Table 4. Other new or changed compiler options

Option/directive	Description
-C!	Specifying the **-C!** compiler option removes comments from preprocessed output.
-qcommon | **-qnocommon**	With **-qcommon** in effect, uninitialized global variables are allocated in the common section of the object file. When **-qnocommon** is in effect, uninitialized global variables are initialized to zero and allocated in the data section of the object file.

Table 4. Other new or changed compiler options (continued)

Option/directive	Description
-qoptdebug | **-qnooptdebug**	When used with optimization levels of **-O3** or higher, the new **-qoptdebug** option instructs the compiler to produce optimized pseudocode that can be read by a symbolic debugger.
-qpack_semantic= ibm | **gnu**	The **-qpack_semantic** option is a portability option that instructs the compiler to use either IBM or GCC syntax and semantics for the **#pragma pack** directive.
-qreport	When used together with compiler options that enable automatic parallelization or vectorization, the **-qreport** option now produces a pseudo-code listing showing how program loops are parallelized and vectorized. The report also provides diagnostic information if the compiler is not able to parallelize or vectorize a given loop.
-qsaveopt | **-qnosaveopt**	In previous releases, the **-qsaveopt** option stored the command line options used to compile a file into the resulting object file. In this release, the information stored in the object file is expanded to also include version and level information for each compiler component invoked during compilation.
-qsmp=stackcheck	This new **-qsmp** suboption instructs the compiler to check for stack overflow by slave threads at run time, and issue a warning if the remaining stack size is less than the number of bytes specified by the **stackckeck** option of the XLSMPOPTS environment variable.
-qtemplatedepth=*number*	**-qtemplatedepth** specifies the maximum number of recursively-instantiated template specializations that the compiler will process.
-qversion=verbose	The **-qversion** option adds a new **verbose** suboption. Specifying **-qversion=verbose** instructs the compiler to display the version and level information for each compiler component invoked during compilation.

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

For complete prerequisite and installation information, refer to the *XL C/C++ Installation Guide*.

Using custom compiler configuration files

A default compiler configuration file is created during XL C/C++ compiler installation, and you can directly modify this configuration file to add default options for specific needs. However, if you later apply updates to the compiler, you will also need to reapply all of your modifications to the newly installed configuration file.

You can avoid this by creating your own custom compiler configuration files. The compiler now has the ability to recognize and resolve compiler settings you specify in your custom configuration files together with compiler settings specified in the default configuration file.

If you instruct the compiler to use a custom configuration file, the compiler will examine and process the settings in that custom configuration file before looking at settings in the default system configuration file. Compiler updates that may later affect settings in the default configuration file will not affect the settings in your custom configuration files.

See "Using custom compiler configuration files" in the *XL C/C++ Compiler Reference* for more information.

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

If contacting software support for assistance, you will need to know what level of XL C/C++ is installed on a particular 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:

```
xlc++ -qversion=verbose
```

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

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

Note:
1. Before you can use the compiler, you must first ensure that XL C/C++ is 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 the following 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. Program assembly for **.s** files and for **.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

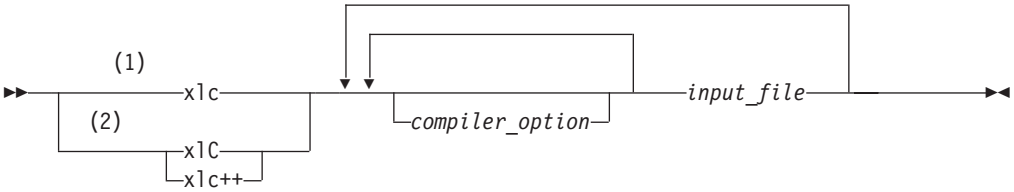
To create C/C++ source programs, you can use any text editor available to your system. Source programs must be saved using a recognized file name suffix. See the "XL C/C++ input and output files" on page 20 for a list of suffixes recognized by XL C/C++.

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++

Invoking the compiler

To compile a source program, use the basic invocation syntax shown below:



Notes:

1 Basic invocation to compile C source code.

2 Basic invocations to compile C++ source code

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.

For new C or C++ application work, you should compile with **xlC**, **xlc++**, or a thread safe counterpart. Both **xlC** and **xlc++** will compile either C or C++ program source, but compiling C++ files with **xlc** 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:

- xlC_r
- xlc++_r
- xlc_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.

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.

Reusing GNU C/C++ compiler options with gxlc and gxlc++

XL C/C++ includes various features to help you transition from GNU C/C++ compilers to XL C/C++. Among these features are the gxlc and gxlc++ utilities.

Each of the gxlc and gxlc++ utilities accepts GNU C or C++ compiler options and translates them into comparable XL C/C++ options. Both utilities use the XL C/C++ options to create an **xlc** or **xlc++** invocation command, which is then used to invoke the compiler. These utilities are provided to help you reuse makefiles

created for applications previously developed with GNU C/C++. However, to fully exploit the capabilities of XL C/C++, you should use the XL C/C++ invocation commands and their associated options.

The actions of gxlc and gxlc++ are controlled by the configuration file `gxlc.cfg`. The GNU C/C++ options that have an XL C/C++ counterpart are shown in this file. Not every GNU option has a corresponding XL C/C++ option. gxlc and gxlc++ return warnings for input options that were not translated.

The gxlc and gxlc++ option mappings are modifiable. For information on using the gxlc or gxlc++ configuration file, see Reusing GNU C/C++ compiler options with gxlc and gxlc++ in the *XL C/C++ Compiler Reference*.

XL C/C++ input and output files

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

Table 5. Input file types

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

Table 6. Output file types

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

Linking your compiled applications with XL C/C++

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

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

compiles and produces 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.

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

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.

  Because this form of linking is the default, you need no additional options to turn it on.
- Static linking means that the code for all routines called by your program becomes part of the executable file.

  Statically linked programs can be moved to and 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.

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++ Programming Guide*

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, enter the name of the program executable file together with any run time arguments on the command line.

You should avoid giving your program executable file the same name as system or shell commands, such as **test** or **cp**, as you could accidentally execute the wrong command. If you do decide to name your program executable file with the same name as a system or shell command, you should execute your program by specifying the path name to the directory in which your program executable file resides, such as `./test`.

Canceling execution

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

To cancel a running program, press the **Ctrl+C** key while the program is in the foreground.

Setting runtime options

You can use environment variable settings to control certain runtime options and behaviors of applications created with the XL C/C++ compiler. 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*.

Running compiled applications on other systems

If you want to run an application developed with the XL C/C++ compiler on another system that does not have the compiler installed, you will need to install a runtime environment on that system.

You can obtain the latest XL C/C++ Runtime Environment PTF images, together with licensing and usage information, from the XL C/C++ Support page at:

www.ibm.com/software/awdtools/xlcpp/support

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.

The XL C/C++ runtime will also issue messages for certain unsupported operations, particularly I/O-related, when you run your application.

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

Specifying the **-g** or **-qlinedebug** compiler options at compile time instructs the XL C/C++ compiler to include debugging information in compiled output.

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 debugging, see "Optimizing your applications" in the *XL C/C++ Programming Guide*.

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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd. Laboratory
8200 Warden Avenue
Markham, Ontario L6G 1C7
Canada

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

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

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

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

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

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

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, 2007. All rights reserved.

Trademarks and service marks

Company, product, or service names identified in the text may be trademarks or service marks of IBM or other companies. Information on the trademarks of International Business Machines Corporation in the United States, other countries, or both is located at http://www.ibm.com/legal/copytrade.shtml.

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

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

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

Industry standards

The following standards are supported:

- The C language is consistent with the International Standard for Information Systems-Programming Language C (ISO/IEC 9899-1990).
- The C language is also consistent with the International Standard for Information Systems-Programming Language C (ISO/IEC 9899-1999 (E)).
- The C++ language is consistent with the International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:1998).
- The C++ language is also consistent with the International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:2003 (E)).
- The C and C++ languages are consistent with the OpenMP C and C++ Application Programming Interface Version 2.5.

Index

Special characters

.a files 20
.c and .C files 20
.i files 20
.lst files 20
.mod files 20
.o files 20
.s files 20
.S files 20
.so files 20

Numerics

64-bit environment 5

A

archive files 20
assembler
  source (.s) files 20
  source (.S) files 20

C

code optimization 5
compilation
  sequence of activities 17
compiler
  controlling behavior of 19
  invoking 18
  running 18
compiler options
  conflicts and incompatibilities 19
  specification methods 19
compiling
  SMP programs 18
customization
  for compatibility with GNU 2

D

debugger support 22
  output listings 22
  symbolic 7
debugging 22
debugging compiled applications 22
debugging information, generating 22
dynamic linking 21

E

editing source files 17
executable files 20
executing a program 21
executing the compiler 18
executing the linker 21

F

files
  editing source 17
  input 20
  output 20

G

GNU
  compatibility with 2

I

input files 20
invoking a program 21
invoking the compiler 18

L

language support 2
level of XL C/C++, determining 15
libraries 20
linking
  dynamic 21
  static 21
linking process 20
listings 20

M

migration
  source code 19
mod files 20
multiprocessor systems 6

O

object files 20
  creating 21
  linking 21
OpenMP 6
optimization
  programs 5
output files 20

P

parallelization 6
performance
  optimizing transformations 5
problem determination 22
programs
  running 21

R

running the compiler 18
runtime
  libraries 20
runtime environment 22
runtime options 22

S

shared memory parallelization 6
shared object files 20
SMP
  programs, compiling 18
SMP programs 6
source files 20
source-level debugging support 7
static linking 21
symbolic debugger support 7

T

tools 4
  cleanpdf utility 4
  configuration file utility 4
  gxlc and gxlc++ utilities 4
  mergepdf utility 4
  new install configuration utility 4
  new_install utility 4
  resetpdf utility 4
  showpdf utility 4
  xlc_configure 4

U

utilities 4
  cleanpdf 4
  gxlc and gxlc++ 4
  mergepdf 4
  new_install 4
  resetpdf 4
  showpdf 4
  xlc_configure 4

V

vac.cfg file 19

X

xlc_configure 4

Program Number: 5724-S73

GC23-5891-00