IBM XL C/C++ for AIX, V10.1

Getting Started with XL C/C++

Version 10.1

GC23-8897-00

IBM XL C/C++ for AIX, V10.1

Getting Started with XL C/C++

Version 10.1

GC23-8897-00

Note

Before using this information and the product it supports, read the information in "Notices" on page 31.

First edition

This edition applies to IBM XL C/C++ for AIX, V10.1 (Program number 5724-U81) 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 AIX®, V10.1 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 **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++ 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 explains the typographical conventions used in the IBM XL C/C++ for AIX, V10.1 information.

Table 1. Typographical conventions

Typeface	Indicates	Example
bold	Lowercase commands, executable names, compiler options, and directives.	The compiler provides basic invocation commands, **xlc** and **xlC** (**xlc++**), along with several other compiler invocation commands to support various C/C++ language levels and compilation environments.
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: `xlc myprogram.c -O3`.

Qualifying elements (icons)

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

Table 2. Qualifying elements

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.

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.

```
►►──keyword──┬──────────────────────┬──────────────────────────────────►◄
             ├─optional_argument1───┤
             └─optional_argument2───┘
```

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

```
             ┌─,──────────────────┐
             ▼                    │
►►──keyword──┴─repeatable_argument─┴──────────────────────────────────►◄
```

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

```
             ┌─default_argument───┐
►►──keyword──┴─alternate_argument─┴──────────────────────────────────►◄
```

- 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.

```
   (1)    (2)        (3)        (4)    (5)                                                  (9)  (10)
►►──────#──────pragma──────comment──────(──┬─compiler──────────────────────────────────┬─)──────────────►◄
                                          ├─date──────────────────────────────────────┤
                                          ├─timestamp─────────────────────────────────┤
                                          │          (6)                              │
                                          │ ┌─copyright─┐   ┌──────────────────────┐  │
                                          └─┴─user──────┴───┴─,──────"──token_sequence──"──┴──┘
                                                              (7)                    (8)
```

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 AIX, V10.1 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 AIX, V10.1 Installation Guide*.

  The information center is viewable on the Web at http://publib.boulder.ibm.com/infocenter/comphelp/v101v121/index.jsp.

- PDF documents

  PDF documents are located by default in the /usr/vacpp/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 information:

Table 3. XL C/C++ PDF files

Document title	PDF file name	Description
IBM XL C/C++ for AIX, V10.1 Installation Guide, GC23-8889-00	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++ for AIX, V10.1, GC23-8897-00	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 AIX, V10.1 Compiler Reference, SC23-8886-00	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 AIX, V10.1 Language Reference, SC23-8888-00	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 AIX, V10.1 Optimization and Programming Guide, SC23-8887-00	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.
Standard C++ Library Reference, SC23-8890-00	standlib.pdf	Contains reference information about the standard C++ runtime libraries and headers.

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

Document title	PDF file name	Description
C/C++ Legacy Class Libraries Reference, SC09-7652-00	legacy.pdf	Contains reference information about the USL I/O Stream Library and the Complex Mathematics Library.

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 information 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

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.
- *AltiVec Technology Programming Interface Manual*, Motorola Inc. This specification for vector data types, to support vector processing technology, is available at http://www.freescale.com/files/32bit/doc/ref_manual/ALTIVECPIM.pdf.
- *Information Technology – Programming Languages – Extension for the programming language C to support decimal floating-point arithmetic, ISO/IEC WDTR 24732*. This draft technical report has been submitted to the C standards committee, and is available at http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1176.pdf.
- *Decimal Types for C++: Draft 4* http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n1977.html

Other IBM information

- *AIX Commands Reference, Volumes 1 - 6, SC23-4888*
- *Technical Reference: Base Operating System and Extensions, Volumes 1 & 2, SC23-4913*

- *AIX National Language Support Guide and Reference*, *SC23-4902*
- *AIX General Programming Concepts: Writing and Debugging Programs*, *SC23-4896*
- *AIX Assembler Language Reference*, *SC23-4923*

  All AIX information is available at http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp.
- *Parallel Environment for AIX: Operation and Use*
- *ESSL for AIX V4.2 Guide and Reference*, *SA22-7904*, available at http://publib.boulder.ibm.com/clresctr/windows/public/esslbooks.html

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/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 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 information or any other XL C/C++ information, send your comments by e-mail 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 AIX, V10.1 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

IBM XL C/C++ for AIX, V10.1 is part of a larger family of IBM C, C++, and Fortran compilers.

XL C/C++, together with XL C and 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/L™, IBM Blue Gene/P™, the Cell Broadband Engine™ architecture, IBM i5/OS®, 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.

Hardware and operating system support

IBM XL C/C++ for AIX, V10.1 supports AIX 5L™ for POWER™ Version 5.3 and AIX Version 6.1. 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 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

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, **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 plain text 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 21 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 V3.0 extensions to support portable parallelized programming
- Language extensions to support vector programming
- A subset of GNU C and C++ language extensions

See "Language standards and specifications documents" 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.

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.

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.

Standard C++ library

XL C/C++ ships a modified version of the Dinkum C++ Library, a conforming implementation of the Standard C++ Library. The Standard C++ Library consists of 51 headers, including 13 headers which constitute the Standard Template Library (STL). In addition, the Standard C++ Library works in conjunction with the 18 headers from the Standard C Library. The functions in these headers perform essential services such as input and output. They also provide efficient implementations of frequently used operations.

For more information, see the *Standard C++ Library Reference*.

C++ library extensions

In addition to the Standard C++ Library, XL C/C++ V10.1 supports extensive extensions to the C++ language as defined by the Draft Technical Report on C++ Library Extensions (TR1).

For more information on these language extensions, see Draft Technical Report on C++ Library Extensions (TR1).

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++ 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*.
- The memory debug runtime library is used for diagnosing memory leaks. See "Using memory heaps" in the *XL C/C++ Optimization and Programming Guide*.
- C++ Runtime Library contains support routines needed by the compiler.
- UNIX® System Laboratories (USL) contains stream classes for input and output capabilities for C++. This library is provided for use by old applications. For new applications, you should use the Standard C++ Library. See *C/C++ Legacy Class Libraries Reference* for more information.
- USL contains classes for manipulating complex numbers. This library is provided for use by old applications. For new applications, you should use the Standard C++ Library. See *C/C++ Legacy Class Libraries Reference* for more information.
- C++ The demangler library provides routines and classes for demangling linkage names created by the C++ compiler.

Support for Boost libraries

In addition to the libraries that ship with XL C/C++, this version of the product supports the Boost V1.34.1 libraries. A patch file is available that modifies the Boost 1.34.1 libraries so that they can be built and used with XL C/C++

applications. The patch or modification file does not extend or otherwise provide additional functionality to the Boost libraries.

To download the patch file and for more information on support for these libraries see the relevant links on the XL C/C++ Library page at http://www.ibm.com/software/awdtools/xlcpp/library.

Tools and utilities

There are many tools and utilities that are included with XL C/C++.

vacppndi
This is a script you can use to install XL C/C++ to a non-default directory location.

IBM Debugger for AIX
The IBM Debugger for AIX can help you detect and diagnose errors in programs that are running locally or remotely. You can control the execution of your programs by setting compiled language-specific breakpoints, suspending execution, stepping through your code, and examining and changing the contents of variables.

The debugger contains views and functionality specific to a given programming language. With the compiled language views, you can monitor variables, expressions, registers, memory, and application modules of the application you are debugging.

CreateExportList command
Creates a file containing a list of all the global symbols found in a given set of object files.

C++ **c++filt Name demangling utility**
When XL C/C++ compiles a C++ program, it encodes all function names and certain other identifiers to include type and scoping information. This encoding process is called mangling. This utility converts the mangled names to their original source code names.

C++ **linkxlC command**
Links C++ .o and .a files. This command is used for linking on systems without XL C/C++ compiler installed.

C++ **makeC++SharedLib command**
Permits the creation of C++ shared libraries on systems on which the XL C/C++ compiler is not installed.

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** command, 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++ for AIX, V10.1 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++ for AIX, V10.1.

Program optimization

XL C/C++ provides several compiler options that can help you control the optimization or performance 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.

For more information, see:
- "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*

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 AIX 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 binder 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
- Attempts to run 64-bit applications on 32-bit platforms

On both 64-bit and 32-bit platforms, 32-bit executables will continue to run as they currently do on a 32-bit platform.

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++ Optimization and 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 (OpenMP, SMP)
- Instructing the compiler to automatically generate shared memory parallelization
- Message passing based shared or distributed memory parallelization (MPI)
- POSIX threads (Pthreads) parallelization
- Low-level UNIX parallelization using `fork()` and `exec()`

The parallel programming facilities of the AIX operating system are based on the concept of threads. Parallel programming exploits the advantages of multiprocessor systems, while maintaining a full binary compatibility with existing uniprocessor systems. This means that a multithreaded program that works on a uniprocessor system can take advantage of a multiprocessor system without recompiling.

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 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.

Beginning with this release, XL C/C++ supports the OpenMP API Version 3.0 specification. See "OpenMP 3.0" on page 11 for an overview of the changes introduced by this feature.

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 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 **dbx**, the IBM Debugger for AIX, or any other symbolic debugger that supports the AIX XCOFF executable format to help you debug your programs.

Chapter 2. What's new for IBM XL C/C++ for AIX, V10.1

This section describes new added features and enhancements in IBM XL C/C++ for AIX, V10.1.

Operating system support

IBM XL C/C++ for AIX, V10.1 now supports AIX V6.1 as well as AIX V5.3.

This version of the compiler does not support AIX V5.2.

C++0x

This release introduces support for a new version of the standard for the C++ programming language - specifically C++0x. This standard has not yet been officially adopted but we are beginning to support some of its features.

Specifically, in this release:

- a new language level has been created
- we introduce new integer promotion rules for arithmetic conversions with `long long` data types
- the C++ preprocessor now supports C99 features

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 in this release. **-qlanglvl=extended0x** is used to allow users to try out early implementations of any features of C++0x that are currently supported by XL C/C++.

C99 long long under C++

With this release of XL C/C++ V10.1, 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 will be promoted to the first type in this list into which it fits:

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

Starting with this release and when compiling with **-qlanglvl=extended0x**, the compiler will now promote 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**.

For additional information about the language standards supported by XL C/C++, see "Language levels and extensions" in the *XL C/C++ Language Reference*.

Other XL C/C++ language-related updates

Vector data types

Vector data types can now use some of the operators that can be used with base data types such as:.

- unary operators
- binary operators
- relational operators

Thread local storage

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`.

OpenMP 3.0

In this release, XL C/C++ supports the OpenMP API Version 3.0 specification. The XL C/C++ implementation is based on IBM's interpretation of the OpenMP Application Program Interface Draft 3.0 Public Comment.

The main differences between Version 2.5 and Version 3.0 are:

- Addition of task level parallelization. The new OpenMP constructs `TASK` and `TASKWAIT` give users the ability to parallelize irregular algorithms, such as pointer chasing or recursive algorithms for which the existing OpenMP constructs were not adequate.
- `for` loops can now contain *var* values of `unsigned int` and `pointer` type as well as `signed int`
- Stack size control. You can now control the size of the stack for threads created by the OMP runtime library using the new environment variable `OMP_STACKSIZE`
- Users can give hints to the desired behavior of waiting threads using new environment variables `OMP_WAIT_POLICY` and `OMP_SET_POLICY`
- Storage reuse. Some restrictions on the `PRIVATE` clause have been removed. A list item that appears in the `reduction` clause of a parallel construct can now also appear in a `private` clause on a work-sharing construct.
- Scheduling. A new `SCHEDULE` attribute, `auto` allows the compiler and runtime system to control scheduling.
- Consecutive loop constructs with `STATIC` schedule can now use `nowait`.
- Nesting support - a `COLLAPSE` clause has been added to the `DO`, `FOR`, `PARALLELL FOR`, and `PARALLEL DO` directives to allow parallelization of perfect loop nests. This means that multiple loops in a nest can be parallelized.
- `THREADPRIVATE` directives can now apply to variables at class scope in addition to file and block scope.
- Parallelization of iterator loops of canonical form including those with random access iterators.

For more information, see:

- "Using OpenMP directives" in the *XL C/C++ Optimization and Programming Guide*
- www.openmp.org

Performance and optimization

Some features and enhancements can assist with performance tuning and optimization of your application.

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 **-qnoqstrict** allowed transformations that could change program semantics. Since higher level of optimizations may require relaxing strict program semantics, the addition of the suboptions allow you to relax selected rules in order to get specific benefits of faster code without turning off all semantic verification.

There are 16 new suboptions that can be used separately or by using a suboption group. The groups are

all Disables all semantics-changing transformations, including those controlled by the other suboptions.

ieeefp Controls whether individual operations conform to IEEE 754 semantics.

order Controls whether or not individual operations can be reordered in a way that may violate program language semantics.

precision
Controls optimizations and transformations that may affect the precision of program results.

exceptions
Controls optimizations and transformations that may affect the runtime exceptions generated by the program.

For detailed information about these suboptions, refer to "-qstrict" in the *XL C/C++ Compiler Reference*.

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 and other performance-related compiler options, refer to "Optimization and tuning options" in the *XL C/C++ Compiler Reference*.

Table 4. Performance-related compiler options and directives

Option/directive	Description
-qstrict	Many new suboptions have been added to give you more control over the relaxation of program semantic rules in order to gain some performance benefits.
-qreport	The listing now contains information about how many streams are created for each loop and which loops cannot be SIMD vectorized due to non-stride-one references. You can use this information to improve the performance of your applications.
-qsmp=omp	When **-qsmp=omp** is in effect, the additional functionality of OpenMP API 3.0 is now available. For more information, see "OpenMP 3.0" on page 11.

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

New or changed compiler options and directives

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 5. New or changed compiler options and directives

Option/directive	Description
-qstrict	Many suboptions have been added to the **-qstrict** option to allow more control over optimizations and transformations that violate strict program semantics. See "Performance and optimization" on page 11 for more information.
-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.
-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.
-qnamemangling	There is a new namemangling scheme for this release.
-qsmp=omp	XL C/C++ now supports some features of OpenMP 3.0. For more information, see "OpenMP 3.0" on page 11.
#pragma init and **#pragma fini**	Programmers can use **#pragma init** and **#pragma fini** to specify a list of functions to run before or after main() or when shared libraries are loaded or unloaded. These functions can be used to do initialization and cleanup. **Note:** A C++ invocation, such as **xlC** or the redistributable tools **linkxlC** or **makeC++SharedLib** must be used at link time.
-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`.
-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.
-qunique	**-qunique** now applies to both C and C++.

Enhancements added in Version 9.0

This section describes new added features and enhancements to the compiler in the previous version, Version 9.0.

C/C++ language-related updates

The default language level for C compilations changed, and new support for extensions to the C and C++ programming languages was introduced.

Default language level changed for C - extc99

The default **-qlanglvl** compiler option setting is **extc99** when invoking the C compiler with the **xlc** invocation. This change allows 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 __STDC_VERSION__ changes from 199409 to 19990.

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

Decimal floating point support for C and C++

Decimal floating point arithmetic offers greater computational performance and precision in business and financial applications where numeric data I/O is usually performed in decimal form. Data conversions from decimal type to binary floating point type and back are avoided, as are inherent rounding errors accumulated during data conversions.

XL C/C++ adds support for decimal floating point arithmetic with the following new compiler options:

Table 6. Decimal floating point compiler options

Option/directive	Description
-qdfp | -qnodfp	Specifying **-qdfp** enables compiler support for decimal floating-point data types and literals. If you specify **-qdfp** when compiling for an architecture that does not support decimal floating-point computation, the compiler will assume **-qfloat=dfpemulate** and enable software emulation of decimal floating-point computations.
-qfloat= dfpemulate | nodfpemulate	Specifying **-qfloat=dfpemulate** instructs the compiler to use software emulation when handling decimal floating point computations. This option can be used with any architecture supported by XL C/C++.
y	There are suboptions specific to decimal floating-point arithmetic for the **y** option to control rounding of constant expressions.

Note: Compiler support for decimal floating point operations requires AIX 5L for POWER version 5.3 with the 5300-06 Technology Level or higher.

For more information, see Extension for the programming language C to support decimal floating-point arithmetic: TR 24732 and Decimal Types for C++: Draft 4.

TR1 library extensions for C++

XL C/C++ Version 9.0 introduced support for numerous extensions to the C++ language as defined by the Draft Technical Report on C++ Library Extensions (TR1).

For more information on these language extensions, see Draft Technical Report on C++ Library Extensions (TR1).

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 settings for -qarch, -qtune

The new default **-qarch** and **-qtune** settings are:

- -qarch=ppc
- -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™.

Important: The change to the **-qarch** default suboption setting can affect the results of floating-point short arithmetic computations in your programs. The **-qarch=com** default used in the previous release of the compiler caused such computations to be performed using double precision instructions followed by rounding. The new **-qarch=ppc** default instructs the compiler to generate code that uses short floating point instructions. The difference in computational method can affect the precision of computational results. To achieve the behavior of the previous **-qarch=com** default, specify the new **-qfloat=nosingle** compiler option when compiling your application.

New support for POWER6 processors

XL C/C++ Version 9.0 expanded 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

Support removed for selected processors

XL C/C++ Version 9.0 removed support for processor architectures not supported by AIX V5.2, such as POWER, POWER2™, and PowerPC 601. As a result, the following **-qarch** and **-qtune** suboption settings are no longer supported.

- -qarch= com | pwr | pwr2 | pwr2s | p2sc | 601 | 602 | 603
- -qtune= pwr | pwr2 | pwr2s | pwrx | p2sc | 601 | 602 | 603

The compiler continues to recognize these suboption settings, and will still generate code for their corresponding architectures. However, in some cases the behavior of that code might differ from code generated by previous versions of the compiler. Also, code generated for these unsupported architectures may not even execute at all on supported AIX systems because of differences in architecture.

Use caution if you will still be using these unsupported **-qarch** and **-qtune** suboption settings.

Performance and optimization

Many enhancements were made to assist with performance tuning and program optimization.

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 7. 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.
-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.
-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= hscmplx|nohscmplx	Specifying **-qfloat=hscmplx** improves optimization of operations involving complex division and complex absolute values.

Table 7. Performance-related compiler options and directives (continued)

Option/directive	Description
-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**.
-qfloat= single | nosingle	Specifying **-qfloat=single** instructs the compiler to compute single-precision floating-point values using single-precision arithmetic instructions supported by all current PowerPC processors. Use **-qfloat=nosingle** if you need to preserve the computational behavior in applications originally compiled for earlier processors, such as POWER and POWER2 processors. You may also need to specify **-qfloat=norndsngl** to obtain the same computational results.
-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.
-qminimaltoc | -qnominimaltoc	Specifying **-qminimaltoc** helps avoid toc overflow conditions in 64-bit compilations by placing toc entries into a separate data section for each object file.
-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++ Optimization and 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.
-qspeculateabsolutes | -qnospeculateabsolutes	During program optimization, **-qnospeculateabsolutes** works with **-qtocmerge -bI:file** for non-IPA links and **-bI:file** for IPA links to disable speculation of variables at absolute addresses.
#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 in Version 9.0

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

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 8. Other new or changed compiler options

Option/directive	Description
-C!	Specifying the **-C!** compiler option removes comments from preprocessed output.
-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.
-qreport	When used together with compiler options that enable automatic parallelization or vectorization, the **-qreport** option 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 Version 9.0, the information stored in the object file 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.

Table 8. Other new or changed compiler options (continued)

Option/directive	Description
-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 "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.

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 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.

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++ 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 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, such as **vi** or **emacs**.

Source programs must be saved using a recognized file name suffix. See the "XL C/C++ input and output files" on page 26 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

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

```
                                                  ┌──────────────────────────┐
                                                  │                          │
                                                  │   ┌──────────────────┐   │
                                                  │   │                  │   │
      (1)                                         │   │                  │   │
 ►►──────────xlc──────────┬──────────────────────┴───┴──┬───────────────┬──┴───input_file──┬──────────►◄
      (2)                 │                             │               │                  │
           ┌──────xlC─────┴─────┐                       └─compiler_option┘                │
           │                    │                                                         │
           └──────xlc++─────────┘                                                         │
```

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 application work, you should compile with **xlc**, **xlc++**, or a thread safe counterpart. You can use **xlc++** to 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_r7, , xlC128_r, xlC128_r7
- xlc++_r, xlc++_r7, xlc++128_r, xlc++128_r7
- xlc_r, xlc_r7, xlc128_r, xlc128_r7

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 SMP or 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.

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++ including gxlc and gxlc++.

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

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 9. Input file types

Filename extension	Description
.a	Archive or library files
.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 files

Table 10. Output file types

Filename extension	Description
a.out	Default name for executable file created by the compiler
.d	Target file suitable for inclusion in a makefile
.i	Preprocessed source files
.lst	Listing files
.o	Object files
.s	Assembler files
.so	Shared object files
.u	Make dependency 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 `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.

```
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
```

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*

Relinking an existing executable file

The linker accepts executable files as input, so you can link an existing executable file with updated object files.

You cannot, however, relink executable files that were previously linked using the **-qipa** option.

If you have a program consisting of several source files and only make localized changes to some of the source files, you do not necessarily have to compile each file again. Instead, you can include the executable file as the last input file when compiling the changed files:

```
xlc -omansion front_door.c entry_hall.c parlor.c sitting_room.c \
  master_bath.c kitchen.c dining_room.c pantry.c utility_room.c

vi kitchen.c # Fix problem in OVEN function

xlc -o newmansion kitchen.c mansion
```

Limiting the number of files to compile and link the second time reduces the compile time, disk activity, and memory use.

Note: You should avoid this type of linking unless you are experienced with linking. If done incorrectly, it can result in interface errors and other problems. If you do encounter problems, compiling with the **-qextchk** compiler option can help you diagnose problems with linking.

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.

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 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

In general, applications linked on a system using an earlier version of AIX will run with more recent versions of AIX. However, applications linked on a system using a newer version of AIX will not necessarily run with earlier versions of AIX.

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.

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++.

Specifying the **-g** or **-qlinedebug** compiler options at compile time instructs the XL C/C++ compiler to include debugging information in compiled output. For more information debugging options, see "Error checking and debugging" in the *XL C/C++ Compiler Reference*.

You can then use **dbx**, the IBM Debugger for AIX, or any other symbolic debugger that supports the AIX XCOFF executable format 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

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
```

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, 2008. All rights reserved.

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A complete and current list of IBM trademarks is available on the Web 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, and/or other countries.

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

Special characters

Numerics

A

B

C

D

E

F

G

I

L

M

O

P

R

S

T

U

V

Program Number: 5724-U81

Printed in USA

GC23-8897-00