IBM XL Fortran for Blue Gene/Q, V14.1

Getting Started with XL Fortran

Version 14.1

GC14-7366-00

IBM XL Fortran for Blue Gene/Q, V14.1

Getting Started with XL Fortran

Version 14.1

GC14-7366-00

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 Fortran for Blue Gene/Q, V14.1 (Program 5799-AH1) 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 Fortran for Blue Gene®/Q, V14.1 compiler.

Who should read this document

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

How to use this document

Throughout this document, the **bgxlf** compiler invocation is 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 Fortran applications using the XL Fortran compiler, it does not include the following topics:

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

Conventions

Typographical conventions

The following table shows the typographical conventions used in the IBM XL Fortran for Blue Gene/Q, V14.1 information.

Table 1. Typographical conventions

Typeface	Indicates	Example
bold	Lowercase commands, executable names, compiler options, and directives.	The compiler provides basic invocation commands, **bgxlf**, along with several other compiler invocation commands to support various Fortran language levels and compilation environments.

Table 1. Typographical conventions (continued)

Typeface	Indicates	Example
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.f, enter: `bgxlf myprogram.f -O3`.
UPPERCASE bold	Fortran programming keywords, statements, directives, and intrinsic procedures. Uppercase letters may also be used to indicate the minimum number of characters required to invoke a compiler option/suboption.	The **ASSERT** directive applies only to the **DO** loop immediately following the directive, and not to any nested **DO** loops.

Qualifying elements (icons and bracket separators)

In descriptions of language elements, this information uses icons and marked bracket separators to delineate the Fortran language standard text as follows:

Table 2. Qualifying elements

Icon	Bracket separator text	Meaning
▸ F2008 F2008 ◂	N/A	The text describes an IBM XL Fortran implementation of the Fortran 2008 standard.
▸ F2003 F2003 ◂	Fortran 2003 begins / ends	The text describes an IBM XL Fortran implementation of the Fortran 2003 standard, and it applies to all later standards.
▸ IBM IBM ◂	IBM extension begins / ends	The text describes a feature that is an IBM XL Fortran extension to the standard language specifications.

Note: If the information is marked with a Fortran language standard icon or bracket separators, it applies to this specific Fortran language standard and all later ones. If it is not marked, it applies to all Fortran language standards.

Syntax diagrams

Throughout this information, diagrams illustrate XL Fortran 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.

IBM XL Fortran extensions are marked by a number in the syntax diagram with an explanatory note immediately following the diagram.

Program units, procedures, constructs, interface blocks and derived-type definitions consist of several individual statements. For such items, a box encloses the syntax representation, and individual syntax diagrams show the required order for the equivalent Fortran statements.

- 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─┘
```

Note: Optional items (not in syntax diagrams) are enclosed by square brackets ([and]). For example, [UNIT=]u

- 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┴────────────────────────────────►◄
```

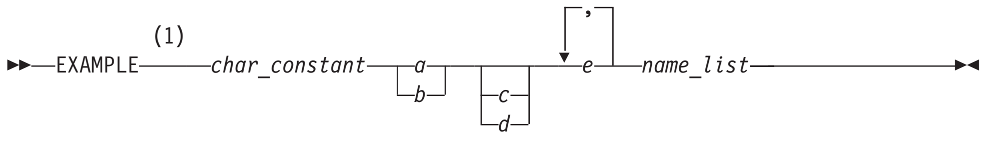
- The item that is the default is shown above the main path.

```
             ┌─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 a variable or user-specified name ends in *_list*, you can provide a list of these terms separated by commas.
- 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 is an example of a syntax diagram with an interpretation:



Notes:

1 IBM extension

Interpret the diagram as follows:
- Enter the keyword EXAMPLE.
- EXAMPLE is an IBM extension.
- Enter a value for *char_constant*.
- Enter a value for *a* or *b*, but not for both.
- Optionally, enter a value for *c* or *d*.
- Enter at least one value for *e*. If you enter more than one value, you must put a comma between each.
- Enter the value of at least one *name* for *name_list*. If you enter more than one value, you must put a comma between each. (The *_list* syntax is equivalent to the previous syntax for *e*.)

How to read syntax statements

Syntax statements are read from left to right:
- Individual required arguments are shown with no special notation.
- When you must make a choice between a set of alternatives, they are enclosed by { and } symbols.
- Optional arguments are enclosed by [and] symbols.
- When you can select from a group of choices, they are separated by | characters.
- Arguments that you can repeat are followed by ellipses (...).

Example of a syntax statement

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

The following list explains the syntax statement:
- Enter the keyword EXAMPLE.
- Enter a value for *char_constant*.
- Enter a value for *a* or *b*, but not for both.

- Optionally, enter a value for *c* or *d*.
- Enter at least one value for *e*. If you enter more than one value, you must put a comma between each.
- Optionally, enter the value of at least one *name* for *name_list*. If you enter more than one value, you must put a comma between each *name*.

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

Examples in this information

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

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

Notes on the terminology used

Some of the terminology in this information is shortened as follows:
- The term *free source form format* often appears as *free source form*.
- The term *fixed source form format* often appears as *fixed source form*.
- The term *XL Fortran* often appears as *XLF*.

Related information

The following sections provide related information for XL Fortran:

IBM XL Fortran information

XL Fortran 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 Fortran 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 Fortran for Blue Gene/Q, V14.1 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 Fortran for Blue Gene/Q, V14.1 Installation Guide*.

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

PDF documents are located by default in the /opt/ibmcmp/xlf/bg/14.1/doc/en_US/pdf/ directory. The PDF files are also available on the web at http://www.ibm.com/software/awdtools/fortran/xlfortran/features/bg/library/.

The following files comprise the full set of XL Fortran product information:

Table 3. XL Fortran PDF files

Document title	PDF file name	Description
IBM XL Fortran for Blue Gene/Q, V14.1 Installation Guide, *GC14-7367-00*	install.pdf	Contains information for installing XL Fortran and configuring your environment for basic compilation and program execution.
Getting Started with IBM XL Fortran for Blue Gene/Q, V14.1, *GC14-7366-00*	getstart.pdf	Contains an introduction to the XL Fortran product, with information on setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
IBM XL Fortran for Blue Gene/Q, V14.1 Compiler Reference, *GC14-7368-00*	compiler.pdf	Contains information about the various compiler options and environment variables.
IBM XL Fortran for Blue Gene/Q, V14.1 Language Reference, *GC14-7369-00*	langref.pdf	Contains information about the Fortran programming language as supported by IBM, including language extensions for portability and conformance to nonproprietary standards, compiler directives and intrinsic procedures.
IBM XL Fortran for Blue Gene/Q, V14.1 Optimization and Programming Guide, *SC14-7370-00*	proguide.pdf	Contains information on advanced programming topics, such as application porting, interlanguage calls, floating-point operations, input/output, application optimization and parallelization, and the XL Fortran high-performance libraries.

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

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

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

Standards and specifications

XL Fortran 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.

- *American National Standard Programming Language FORTRAN, ANSI X3.9-1978.*
- *American National Standard Programming Language Fortran 90, ANSI X3.198-1992.*
- *ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985.*
- *Federal (USA) Information Processing Standards Publication Fortran, FIPS PUB 69-1.*
- *Information technology - Programming languages - Fortran, ISO/IEC 1539-1:1991 (E).* (This information uses its informal name, Fortran 90.)
- *Information technology - Programming languages - Fortran - Part 1: Base language, ISO/IEC 1539-1:1997.* (This information uses its informal name, Fortran 95.)

- *Information technology - Programming languages - Fortran - Part 1: Base language, ISO/IEC 1539-1:2004*. (This information uses its informal name, Fortran 2003.)
- *Information technology - Programming languages - Fortran - Part 1: Base language, ISO/IEC 1539-1:2010*. (This information uses its informal name, Fortran 2008.)
- *Military Standard Fortran DOD Supplement to ANSI X3.9-1978, MIL-STD-1753* (United States of America, Department of Defense standard). Note that XL Fortran supports only those extensions documented in this standard that have also been subsequently incorporated into the Fortran 90 standard.
- *OpenMP Application Program Interface Version 3.1*, available at http://www.openmp.org

Other IBM information

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

Technical support

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

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

For the latest information about XL Fortran, visit the product information site at http://www.ibm.com/software/awdtools/fortran/xlfortran/features/bg/.

How to send your comments

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

Be sure to include the name of the information, the part number of the information, the version of XL Fortran, 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 Fortran

IBM XL Fortran for Blue Gene/Q, V14.1 is an advanced, high-performance compiler that can be used for developing complex, computationally intensive programs, including interlanguage calls with C and C++ programs.

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

About the Blue Gene architecture

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

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

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

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

Commonality with other IBM compilers

IBM XL Fortran for Blue Gene/Q, V14.1 is part of a larger family of IBM C, C++, and Fortran compilers.

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

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

Operating system support

IBM XL Fortran for Blue Gene/Q, V14.1, along with the compiler related tools, supports the Red Hat Enterprise Linux 6.2 (RHEL 6.2) operating system on Blue Gene/Q Front End nodes.

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

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

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

A highly configurable compiler

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

Compiler invocation commands

XL Fortran provides several different commands that you can use to invoke the compiler, for example, **bgxlf**, **bgxlf90**, **bgxlf95**, **bgxlf2003**, and **bgxlf2008**. 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 Fortran language levels, and many popular language extensions as well.

The compiler also provides corresponding "**_r**" versions of most invocation commands, for example, **bgxlf_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 Fortran compiler invocation commands, see "Compiling XL Fortran programs" in the *XL Fortran 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 Fortran compilers, and perform many other common tasks that would otherwise require changing the source code.

XL Fortran 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 Fortran compiler options, see "Summary of compiler options" in the *XL Fortran Compiler Reference*.

Custom compiler configuration files

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

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

For more information about using custom compiler configuration files, see "Using custom compiler configuration files" on page 21.

Language standard compliance

This section provides language standard compliance information for IBM XL Fortran for Blue Gene/Q, V14.1.

The compiler supports the following programming language specifications for Fortran:

- ANSI X3.9-1978 (referred to as FORTRAN 77)
- ISO/IEC 1539-1:1991(E) and ANSI X3.198-1992 (referred to as Fortran 90 or F90)
- ISO/IEC 1539-1:1997 (referred to as Fortran 95 or F95)
- ISO/IEC 1539-1:2004 (referred to as Fortran 2003 or F2003)
- Partial support for ISO/IEC 1539-1:2010 (referred to as Fortran 2008 or F2008)

In addition to the standardized language levels, XL Fortran supports language extensions, including:

- OpenMP Application Program Interface V3.1
- Language extensions to support vector programming
- Common Fortran language extensions defined by other compiler vendors, in addition to those defined by IBM
- Industry extensions that are found in Fortran products from various compiler vendors
- Extensions specified in SAA Fortran

See "Language standards" in the *XL Fortran Language Reference* for more information about Fortran language specifications and extensions.

Source-code migration and conformance checking

XL Fortran helps protect your investment in your existing Fortran source code by providing compiler invocation commands that instruct the compiler to inspect your application for conformance to a specific language level and warn you if constructs and keywords are found that do not conform to the specified language level.

You can also use the **-qlanglvl** compiler option to specify a given language level, and the compiler will issue warnings if language elements in your program source do not conform to that language level. Additionally, you can name your source files with common filename extensions such as .f77, .f90, f95, .f03, or .f08, then use the generic compiler invocations such as **bgxlf** or **bgxlf_r** to automatically select the appropriate language level appropriate to the filename extension.

See -qlanglvl in the *XL Fortran Compiler Reference* for more information.

Tools, utilities, and commands

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

Utilities

new_install
: The **new_install** utility configures IBM XL Fortran for Blue Gene/Q, V14.1 for use on your system, after you install the compiler.

xlf_configure
: The **xlf_configure** utility creates custom compiler configuration files containing your own custom sets of compiler option default settings. For more information, see Running the xlf_configure utility directly (for advanced users) in the *XL Fortran Installation Guide*.

Commands

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

Program optimization

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

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

Optimizing transformations can give your application better overall execution performance. XL Fortran provides a portfolio of optimizing transformations tailored to various supported hardware. These transformations offer the following benefits:
- Reducing the number of instructions executed for critical operations
- Restructuring generated object code to make optimal use of the Blue Gene architecture
- Improving the usage of the memory subsystem
- Exploiting the ability of the architecture to handle large amounts of shared memory parallelization

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

Shared memory parallelization

XL Fortran supports application development for multiprocessor system architectures.

You can use any of the following methods to develop your parallelized applications with XL Fortran:

- 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 "Parallel programming with XL Fortran" in the *XL Fortran Optimization and Programming Guide*.

OpenMP directives

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

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

OpenMP directives address three important issues of parallelizing an application:

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

As of XL Fortran for Blue Gene/Q, V14.1, XL Fortran supports all features of the OpenMP API Version 3.1 specification. See “OpenMP 3.1” on page 16 for an overview of the support provided by this feature.

Speculative execution of threads

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

For details, see Thread-level speculative execution.

Transactional memory

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

For details, see Transactional memory.

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

Diagnostic listings

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

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

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

Symbolic debugger support

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

For details, see **-g** or **-qdbg** in *XL Fortran Compiler Reference*.

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

Chapter 2. What's new for IBM XL Fortran for Blue Gene/Q, V14.1

This section describes features and enhancements added to the compiler in IBM XL Fortran for Blue Gene/Q, V14.1.

The XL compiler for Blue Gene/Q contains some significant enhancements since XL Fortran, V11.1 that supported Blue Gene/L and Blue Gene/P. It also contains many enhancements that are in common with the compilers on the AIX and Linux platforms.

Blue Gene/Q features

This section describes the Blue Gene/Q new features added to the compiler in IBM XL Fortran for Blue Gene/Q, V14.1.

Quad Processing Extension support

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

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

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

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

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

For more information about the QPX data types and intrinsic procedures, see Vector in the and Vector intrinsic procedures in the *XL Fortran Language Reference*.

Speculative execution of threads

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

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

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

Related information

- "-qsmp"
- Thread-level speculative execution
- SPECULATIVE DO / END SPECULATIVE DO
- SPECULATIVE SECTIONS
- Routines for thread-level speculative execution
- Environment variables for thread-level speculative execution

Transactional memory

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

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

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

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

Related information

- Transactional memory
- TM_ATOMIC
- Routines for transactional memory
- Environment variables for transactional memory

Compiler options and directives support

This section describes new and changed compiler options and directives for IBM XL Fortran for Blue Gene/Q, V14.1.

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

Debug optimized program using -g

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

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

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

For details, see -g or -qdbg.

Compiler options or directives for Blue Gene/Q

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

Blue Gene/Q specific compiler options

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

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

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

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

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

-qflttrap
: **-qflttrap=qpxstore** detects and traps on not-a-number (NaN) or infinity values in Quad Processing eXtension (QPX) vectors. The exceptions only occur on QPX store instructions.

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

Blue Gene/Q specific directives

SPECULATIVE DO / END SPECULATIVE DO
: The **SPECULATIVE DO** directive instructs the compiler to speculatively parallelize a **DO** loop.

SPECULATIVE SECTIONS / END SPECULATIVE SECTIONS
: The **SPECULATIVE SECTIONS** directive instructs the compiler to speculatively parallelize sections of the code. In code blocks delimited by **SPECULATIVE SECTIONS** and **END SPECULATIVE SECTIONS**, you can use the **SPECULATIVE SECTION** directive to delimit program code segments.

TM_ATOMIC / END TM_ATOMIC
: The **TM_ATOMIC** directive indicates a transactional atomic region.

Other new or changed compiler options and directives

This section summarizes other new or changed compiler options and directives since V11.1 of the XL Fortran compiler.

New or changed compiler options

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

-qassert

New suboptions have been added for **-qassert** to provide more information about the characteristics of the files that can help to fine-tune optimizations.

The **-qassert=contig** option tells the compiler that all array pointers are associated with contiguous targets, and that all assumed-shape arrays are associated with contiguous actual arguments.

The **-qassert=refalign** option tells the compiler that all pointers only point to naturally-aligned data.**-qassert=minitercnt=***n* and **-qassert=maxitercnt=***n* are added to specify the expected minimum and maximum iteration counts of the loops in the program.

-qbindcextname
Controls whether the **-qextname** option affects **BIND(C)** entities.

-qfunctrace
Inserts calls to user-defined tracing procedures at procedure entry and exit. You can specify module procedures and module names.

-qfunctrace_xlf_catch, **-qfunctrace_xlf_enter**, and **-qfunctrace_xlf_exit** specify the name of the catch, entry, and exit tracing subroutines.

-qhaltonmsg
Stops compilation before producing any object files, executable files, or assembler source files if a specified error message is generated.

-qhot

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

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

-qinitalloc
The new option **-qinitalloc** is added to initialize allocatable and pointer variables that are allocated but not initialized.

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

-qlanglvl
The following suboptions are added or updated:

-qlanglvl=2008std
-qlanglvl=2008pure
These two new suboptions are added to enable language level checking for supported Fortran 2008 features.

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

-qlistfmt
The **-qlistfmt** option is enhanced to generate HTML reports as well as XML reports, containing information about optimizations performed by the compiler and missed optimization opportunities.

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

-qmaxerr
: **-qmaxerr** stops compilation when the number of error messages of a specified severity level or higher reaches a specified number.

-qmkshrobj
: Creates a shared object from generated object files.

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

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

-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.

-qsmp=omp
: When **-qsmp=omp** is in effect, some of the additional functionality of OpenMP API 3.1 is now available. For more information, see "OpenMP 3.1" on page 16.

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

-qstrict
: Many suboptions have been added to the **-qstrict** option to allow more control over optimizations and transformations that violate strict program semantics. For details, see "Performance and optimization" on page 16.

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

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

-qxlf2008
: The new suboption **-qxlf2008=checkpresence** is added so that you can check dummy argument presence according to the Fortran 2008 standard.

-qxlf2003
: The new suboption **-qxlf2003=dynamicacval** is added to control whether you can use unlimited polymorphic entities for array constructors, and whether dynamic types of array constructor values are used.

New or changed directives

ALIGN
: Using the ALIGN directive, you can specify the alignment for your variables in memory.

ASSERT
: You can use assertions **MINITERCNT***(n)* and **MAXITERCNT***(n)* to specify the minimum and maximum number of iterations for a given loop.

EXECUTION_FREQUENCY
The `EXECUTION_FREQUENCY` directive marks source code that you expect will be executed very frequently or very infrequently.

IGNORE_TKR
The `IGNORE_TKR` directive facilitates portability of code written for other compilers. It simplifies the writing of generic interfaces, especially for system libraries by directing the compiler to ignore type, kind and rank of dummy arguments.

Language support enhancements

This section describes language enhancements added to the compiler in IBM XL Fortran for Blue Gene/Q, V14.1.

Fortran 2008 features

XL Fortran implements selected features of the Fortran 2008 standard.

This version of XL Fortran provides support for the following Fortran 2008 features:

- **ALLOCATE** enhancements
- Complex part designators
- Implied-shape arrays
- Internal procedures as actual arguments or procedure pointer targets
- Intrinsic types in the **TYPE()** type specifier
- Pointer dummy argument enhancement
- The declaration of multiple type-bound procedures in a single procedure statement
- The **-qxlf2008=checkpresence** suboption
- The **BLOCK** construct
- The **CONTIGUOUS** attribute and **IS_CONTIGUOUS** intrinsic function
- The **END** statement for internal and module subprograms
- The **EXIT** statement
- The **HYPOT** intrinsic procedure
- The **ISO_FORTRAN_ENV** intrinsic module
- The **LEADZ** and **TRAILZ** intrinsic procedures
- The math intrinsic procedures extension
- The **NEWUNIT=** specifier
- The **POPCNT** and **POPPAR** inquiry intrinsic functions
- The **RADIX=** argument
- The **STOP** and **ERROR STOP** statements

ALLOCATE enhancements

The **MOLD=** specifier has been added to the **ALLOCATE** statement. In addition, you can omit the bounds in the **ALLOCATE** statement if you provide *source_expr* in the **SOURCE=** or **MOLD=** specifier.

Complex part designators

Complex part designators have been added in Fortran 2008. Using complex part designators, you can directly access the real or imaginary part of complex entities. You can use the designators instead of the REAL() and IMAG() intrinsics.

Implied-shape arrays

Implied-shape arrays have been added in Fortran 2008. An implied-shape array inherits its shape from the constant expression in its declaration.

Internal procedures as actual arguments or procedure pointer targets

To conform with the Fortran 2008 standard, procedure pointers can now point to internal procedures. In addition, you can use internal procedures and pointers to internal procedures as actual arguments.

Intrinsic types in the TYPE() type specifier

The **TYPE()** type specifier has been extended to declare entities of both derived type and intrinsic type.

Pointer dummy argument enhancement

In Fortran 2008, a dummy argument that has the POINTER and INTENT(IN) attributes can be argument associated with a nonpointer actual argument that has the TARGET attribute.

The declaration of multiple type-bound procedures in a single procedure statement

In Fortran 2008, you can declare multiple type-bound procedures using one type-bound procedure statement.

The -qxlf2008=checkpresence suboption

The **-qxlf2008=checkpresence** suboption has been introduced to check the allocation status or pointer association status of actual arguments during argument association of optional dummy arguments.

The BLOCK construct

The **BLOCK** construct has been added in Fortran 2008. It defines an executable block that can contain declarations.

The CONTIGUOUS attribute and IS_CONTIGUOUS intrinsic function

The **CONTIGUOUS** attribute specifies that the array elements in an array pointer or an assumed-shape array are not separated by other data objects, which guarantees that the array object is stored in contiguous memory.

The **IS_CONTIGUOUS** intrinsic function is used to test whether an array is stored in contiguous memory.

The END statement for internal and module subprograms

In Fortran 2008, you can omit the **FUNCTION** and **SUBROUTINE** keywords on the **END** statements for internal and module subprograms.

The EXIT statement

The **EXIT** statement can now be used to terminate execution of one of the following constructs:

- ASSOCIATE
- BLOCK
- DO
- IF
- SELECT CASE
- SELECT TYPE

The HYPOT intrinsic procedure

The **HYPOT** intrinsic procedure is introduced to calculate the Euclidean distance between two values.

The ISO_FORTRAN_ENV intrinsic module

The following constants are added:

- **CHARACTER_KINDS**
- **INT8**, **INT16**, **INT32**, and **INT64**
- **INTEGER_KINDS**
- **IOSTAT_INQUIRE_INTERNAL_UNIT**
- **LOGICAL_KINDS**
- **REAL32**, **REAL64**, and **REAL128**
- **REAL_KINDS**

The following functions are added:

- **COMPILER_OPTIONS**
- **COMPILER_VERSION**

The LEADZ and TRAILZ intrinsic procedures

The **LEADZ** and **TRAILZ** intrinsic procedures are introduced to count the number of leading and trailing zeros in an integer.

The math intrinsic procedures extension

The following new intrinsic procedures have been introduced:

- **ACOSH**
- **ASINH**
- **ATANH**
- **ERFC_SCALED**
- **LOG_GAMMA**

Notes:

1. The **LOG_GAMMA** intrinsic procedure is the Fortran 2008 standard compliant alias of the **LGAMMA** intrinsic procedure.
2. The **ERF**, **ERFC**, and **GAMMA** intrinsic procedures are now Fortran 2008 standard compliant.

Complex arguments are now supported in the following intrinsic procedures:
- **ACOS**
- **ASIN**
- **ATAN**
- **COSH**
- **SINH**
- **TAN**
- **TANH**

Note: The **ATAN** intrinsic procedure can now optionally take two arguments, **ATAN(Y, X)**, and have the same results as the **ATAN2** intrinsic procedure.

The NEWUNIT= specifier

The **OPEN** statement has been updated with the **NEWUNIT=** specifier to specify the unit number automatically. In the **BACKSPACE**, **CLOSE**, **ENDFILE**, **FLUSH**, **INQUIRE**, **OPEN**, **READ**, **REWIND**, and **WRITE** statements, the range of unit values now includes the NEWUNIT value.

The POPCNT and POPPAR inquiry intrinsic functions

The **POPCNT** and **POPPAR** functions have been updated to conform with the Fortran 2008 standard. They can be used in constant expressions now.

The RADIX= argument

A **RADIX=** argument has been added to the **SELECTED_REAL_KIND** and **IEEE_SELECTED_REAL_KIND** intrinsic procedures.

The STOP and ERROR STOP statements

The **STOP** statement has been enhanced to take an integer or character constant expression as stop code. The **STOP** statement initiates normal termination of a program while the **ERROR STOP** statement initiates error termination.

Other XL Fortran language-related updates

XL Fortran fully implements the Fortran 2003 standard.

Fortran 2003 compliance

XL Fortran fully implements the Fortran 2003 standard. This version of XL Fortran provides the following features:
- Support for parameterized derived types, including kind and length parameters.
- Support for generic interfaces with the same name as derived types.
- The `OPEN` and `INQUIRE` statements have been updated with the `ENCODING=` specifier to indicate the encoding form of the file.

For more information, see Fortran 2003.

IEEE module enhancements

The IEEE_ARITHMETIC module defines a new constant, `IEEE_OTHER_VALUE`, and three new functions:

- `IEEE_SET_UNDERFLOW_MODE`
- `IEEE_GET_UNDERFLOW_MODE`
- `IEEE_SUPPORT_UNDERFLOW_MODE`

OpenMP 3.1

IBM XL Fortran for Blue Gene/Q, V14.1 supports the OpenMP Application Program Interface Version 3.1 specification. The XL Fortran implementation is based on IBM's interpretation of the OpenMP Application Program Interface Version 3.1.

OpenMP 3.1 includes the following updates to OpenMP 3.0:

- Adds `FINAL` and `MERGEABLE` clauses to the `TASK` construct to support optimization.
- Adds the `TASKYIELD` construct to allow users to specify where in the program can perform task switching.
- Adds the `omp_in_final` runtime library routine to support specialization of final task regions.
- Extends the `ATOMIC` construct to include `READ`, `WRITE`, and `CAPTURE` forms; adds the `UPDATE` clause to apply the existing form of the `ATOMIC` construct.
- Allows dummy arguments with the `INTENT(IN)` attribute to be specified on the `FIRSTPRIVATE` clause.
- Allows unallocated allocatable arrays to be specified on the `COPYIN` clause.
- Allows Fortran 90 Pointers to be specified on the `FIRSTPRIVATE` clause.
- Adds the `OMP_PROC_BIND` environment variable to control whether OpenMP threads are allowed to move between processors.
- Extends the `OMP_NUM_THREADS` environment variable to specify the number of threads to use for nested parallel regions.

Related information

- "Parallel programming with XL Fortran" in the *XL Fortran Optimization and Programming Guide*
- www.openmp.org

Performance and optimization

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

Enhancements to -qstrict

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

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

Reports about compiler optimizations

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

Performance-related compiler options and directives

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

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

Table 4. Performance-related compiler options and directives

-qinline=level=*number*	A new option is added to -qinline to provide guidance to the compiler about the relative value of inlining in relation to the default value of 5.*number* is a range of integer values 0 - 10 that indicates the level of inlining you want to use. For details, see -qinline in the *XL Fortran Compiler Reference*.
-qfloat	Some **-qfloat** suboptions are affected by the new suboptions for **-qstrict**.
EXECUTION_FREQUENCY	The `EXECUTION_FREQUENCY` directive marks source code that might be executed very frequently or very infrequently. When optimization is enabled, the directive is used as a hint to the optimizer.

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

New diagnostic reports

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

Compiler reports in XML or HTML format

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

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

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

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

Report of data reorganization

The compiler can generate the following information in the listing files:

- Data reorganizations (a summary of how program variable data gets reorganized by the compiler)
- The location of data prefetch instructions inserted by the compiler

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

- common block splitting
- array splitting
- array transposing
- memory allocation merging
- array interleaving
- array coalescing

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

Additional loop analysis

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

New and enhanced diagnostic options

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

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

Table 5. Listings-related compiler options and directives

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

Chapter 3. Setting up and customizing XL Fortran

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

Using custom compiler configuration files

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

You have the following options to customize compiler settings:

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

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

Chapter 4. Developing applications with XL Fortran

Fortran 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 Fortran and the Blue Gene/Q tool chain are properly installed and configured. For more information, see the *XL Fortran Installation Guide*.
2. To learn about writing Fortran programs, refer to the *XL Fortran 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. Loop transformations
   c. High-level optimization
   d. Low-level optimization
   e. Register allocation
   f. 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 Fortran source files

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

Source programs must be saved using a recognized file name suffix. See “XL Fortran input and output files” on page 28 for a list of suffixes recognized by XL Fortran.

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

Compiling with XL Fortran

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

Compiling Fortran 2008 programs

The Fortran 2008 language standard is partially supported in this release. Use the following invocations (or their variants) to conform more closely to their corresponding Fortran language standards:

Fortran 2008
bgf2008, bgxlf2008, bgxlf2008_r

These compiler invocations accept Fortran 90 free source form by default. To use fixed source form with these invocations, you must specify the **-qfixed** command-line option.

By default, these invocation commands do not conform completely to the Fortran 2008 language standard supported in this release. If you need full compliance, compile with the following additional compiler option settings:

```
-qlanglvl=2008std -qnodirective -qnoescape -qextname
-qfloat=nomaf:nofold -qnoswapomp -qstrictieeemod
```

Also, specify the following run time options before running the program, with a command similar to the following:

```
export XLFRTEOPTS="err_recovery=no:langlvl=2008std:
                   iostat_end=2003std:internal_nldelim=2003std"
```

The default settings are intended to provide the best combination of performance and usability, so you might change them only when absolutely required. Some of the options mentioned above are only required for compliance in very specific situations. For example, you might need to specify **-qextname** only when an external symbol, such as a common block or subprogram, is named **main**.

Compiling Fortran 2003 programs

Use the following invocations (or their variants) to conform more closely to their corresponding Fortran language standards:

Fortran 2003
bgf2003, bgxlf2003, bgxlf2003_r

These compiler invocations are the preferred compiler invocation commands that you should use when creating and compiling new applications.

They accept Fortran 90 free source form by default. To use fixed source form with these invocations, you must specify the **-qfixed** command-line option.

By default, these invocation commands do not conform completely to the Fortran 2003 language standard. If you need full compliance, compile with the following additional compiler option settings:

```
-qlanglvl=2003std -qnodirective -qnoescape -qextname
-qfloat=nomaf:nofold -qnoswapomp -qstrictieeemod
```

Also, specify the following run time options before running the program, with a command similar to the following:

```
export XLFRTEOPTS="err_recovery=no:langlvl=2003std:
                   iostat_end=2003std:internal_nldelim=2003std"
```

The default settings are intended to provide the best combination of performance and usability, so you should change them only when absolutely required. Some of the options mentioned above are only required for compliance in very specific situations. For example, you would need to specify **-qextname** only when an external symbol, such as a common block or subprogram, is named **main**.

-qxlf2003 compiler option

The **-qxlf2003** compiler option provides compatibility with XL Fortran V10.1 and the Fortran 2003 standard for certain aspects of the language.

When compiling with the Fortran 2003 or Fortran 2008 compiler invocations, the default setting is **-qxlf2003=polymorphic**. This setting instructs the compiler to allow polymorphic items such as the CLASS type specifier and SELECT TYPE construct in your Fortran application source.

For all other compiler invocations, the default is **-qxlf2003=nopolymorphic**.

Compiling Fortran 95, or Fortran 90 programs

Use the following invocations (or their variants) to conform more closely to their corresponding Fortran language standards:

Fortran 95
: bgf95, bgxlf95, bgxlf95_r

Fortran 90
: bgf90, bgxlf90, bgxlf90_r

These compiler invocations accept Fortran 90 free source form by default. To use fixed source form with these invocations, you must specify the **-qfixed** command-line option.

I/O formats are slightly different between these commands and the other commands. I/O formats for the Fortran 95 compiler invocations are also different from those of Fortran 90 invocations. We recommend that you switch to the Fortran 95 formats for data files whenever possible.

By default, these invocation commands do not conform completely to their corresponding Fortran language standards. If you need full compliance, compile with the following additional compiler option settings:

For full Fortran 90 compliance:

```
-qlanglvl=90std -qnodirective -qnoescape -qextname
-qfloat=nomaf:nofold -qnoswapomp
```

For full Fortran 95 compliance:

```
-qlanglvl=95std -qnodirective -qnoescape -qextname
-qfloat=nomaf:nofold -qnoswapomp
```

Also, specify the following runtime options before running the program, with a command similar to the following:

For full Fortran 90 compliance:

```
export XLFRTEOPTS="err_recovery=no:langlvl=90std"
```

For full Fortran 95 compliance:

```
export XLFRTEOPTS="err_recovery=no:langlvl=95std"
```

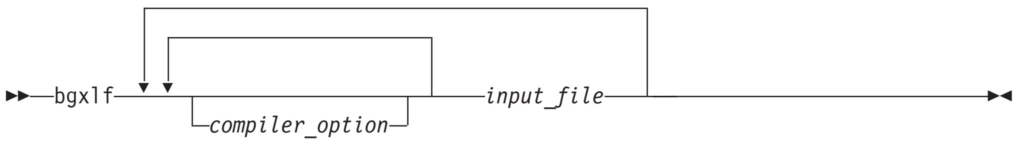
The default settings are intended to provide the best combination of performance and usability, so you should change them only when absolutely required. Some of the options mentioned above are only required for compliance in very specific situations. For example, you would need to specify **-qextname** only when an external symbol, such as a common block or subprogram, is named **main**.

Invoking the compiler

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

To compile a source program, use any of the available XL Fortran for Blue Gene/Q compiler invocation commands. The **bg**-prefixed invocation commands on the Blue Gene/Q Front End node (RHEL 6.2) are for cross-compiling applications for use on the Blue Gene/Q compute node. The invocation commands that are not prefixed with **bg** create executable programs targeted for RHEL 6.2 on POWER® platforms, and are provided only for testing and debugging purposes. For the development of applications targeted for the Front End node, IBM provides the IBM XL Fortran for Linux, V14.1 product. As well, only the compiler options which are supported by the **bg** cross-compiler commands are supported when using these compiler invocations to create executable files for Blue Gene/Q.

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



For most applications, you should compile with **bgxlf** or a thread safe counterpart.

Additional invocation commands are available to meet specialized compilation needs, primarily to provide explicit compilation support for different levels and extensions of the Fortran language. See "Compiling XL Fortran programs" in the *XL Fortran Compiler Reference* for more information about compiler invocation commands available to you.

When working with source files whose filename extensions indicates a specific level of Fortran, such as .f08, .f03, .f95, .f90, or .f77, compiling with **bgxlf**, or corresponding generic thread safe invocations will cause the compiler to automatically select the appropriate language-level defaults.

Compiling parallelized XL Fortran applications

XL Fortran 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 Fortran thread-safe compiler invocation is:

- bgxlf_r

XL Fortran provides additional thread-safe invocations to meet specific compilation requirements. See "Compiling XL Fortran programs" in the *XL Fortran Compiler Reference* for more information.

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

For more information on parallelized applications, see "Parallel programming" in the *XL Fortran Optimization and Programming Guide*.

POSIX Pthreads API support

XL Fortran supports thread programming with the IEEE 1003.1-2001 (POSIX) standard Pthreads API.

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 xlf.cfg file before it searches the directories specified with **-I** on the command-line. The option is cumulative rather than preemptive.

See the *XL Fortran 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 "Specifying options on the command line" in the *XL Fortran Compiler Reference* for more information about compiler options and how to specify them.

XL Fortran input and output files

These file types are recognized by XL Fortran.

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

Table 6. Input file types

Filename extension	Description
.f, .F, .f77, .F77, .f90, .F90, .f95, .F95, .f03, .F03, .f08, .F08	Fortran source files
.mod	Module symbol files
.o	Object files
.s	Assembler files
.so	Shared object or library files

Table 7. Output file types

Filename extension	Description
a.out	Default name for executable file created by the compiler
.mod	Module symbol files
.lst	Listing files
.o	Object files
.s	Assembler files
.so	Shared object or library files

Linking your compiled applications with XL Fortran

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

For example, running the following command:

```
bgxlf file1.f file2.o file3.f
```

compiles `file1.f` and `file3.f` 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.

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

For more information about compiling and linking your programs, see "Linking XL Fortran programs" in the *XL Fortran Compiler Reference*.

Dynamic and static linking

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

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

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

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

Related information

- The -qstaticlink compiler option

Running your compiled application

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

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

Canceling execution

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

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

Setting runtime options

You can use environment variable settings to control certain runtime options and behaviors of applications created with the XL Fortran 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 Fortran Installation Guide*.

XL Fortran compiler diagnostic aids

XL Fortran 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 Fortran Compiler Reference*:
- "Understanding XL Fortran compiler 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 Fortran.

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

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

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

Determining what level of XL Fortran is installed

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

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

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

```
bgxlf -qversion=verbose
```

Notices

Trademarks and service marks

Index

Product Number: 5799-AH1

Printed in USA

GC14-7366-00