

IBM XL Fortran for AIX, V13.1



Getting Started with XL Fortran

Version 13.1

IBM XL Fortran for AIX, V13.1



Getting Started with XL Fortran

Version 13.1

Note

Before using this information and the product it supports, read the information in "Notices" on page 41.

First edition

This edition applies to IBM XL Fortran for AIX, V13.1 (Program 5724-X15) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

© Copyright IBM Corporation 1996, 2010.

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

Contents

About this document	v
Conventions	v
Related information	ix
IBM XL Fortran information	ix
Standards and specifications	x
Other IBM information	xi
Technical support	xi
How to send your comments	xi

Chapter 1. Introducing XL Fortran	1
Commonality with other IBM compilers	1
Hardware and operating system support	1
A highly configurable compiler	1
Language standards compliance	2
Source-code migration and conformance checking	3
Tools and utilities	3
Program optimization	4
64-bit object capability	4
Shared memory parallelization	5
Diagnostic listings	6
Symbolic debugger support	6

Chapter 2. What's new for IBM XL Fortran for AIX, V13.1	7
Support for POWER7 processors	7
XL Fortran language-related updates	8
OpenMP 3.0	8
Performance and optimization	9
New diagnostic reports	11
Utilization tracking and reporting	13
New or changed compiler options and directives	14
Directives and intrinsics new for this release	16
Compatibility of redistributable library libxlopt.a	17

Chapter 3. Enhancements added in previous versions	19
Enhancements added in Version 12.1	19
Operating system support	19
XL Fortran language-related updates	19

OpenMP 3.0	19
Performance and optimization	20
New or changed compiler options and directives	21
Enhancements added in Version 11.1	22
Enhanced support for Fortran 2003	22
Architecture and processor support	24
Performance and optimization	25
Other new or changed compiler options	27

Chapter 4. Setting up and customizing XL Fortran	29
Upgrading to XL Fortran V13.1	29
Using custom compiler configuration files	29
Configuring compiler utilization tracking and reporting	29

Chapter 5. Developing applications with XL Fortran	31
The compiler phases	31
Editing Fortran source files	31
Compiling with XL Fortran	32
Invoking the compiler	33
Compiling parallelized XL Fortran applications	34
Specifying compiler options	35
XL Fortran input and output files	35
Linking your compiled applications with XL Fortran	36
Linking new objects with existing ones	37
Relinking an existing executable file	37
Dynamic and static linking	37
Running your compiled application	38
XL Fortran compiler diagnostic aids	39
Debugging compiled applications	39
Determining what level of XL Fortran is installed	39

Notices	41
Trademarks and service marks	43

Index	45
------------------------	-----------

About this document

This document contains overview and basic usage information for the IBM® XL Fortran for AIX®, V13.1 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 `xlf` 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 explains the typographical conventions used in the IBM XL Fortran for AIX, V13.1 information.

Table 1. *Typographical conventions*

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







Table 1. Typographical conventions (continued)

Typeface	Indicates	Example
<i>italics</i>	Parameters or variables whose actual names or values are to be supplied by the user. Italics are also used to introduce new terms.	Make sure that you update the <i>size</i> parameter if you return more than the <i>size</i> requested.
<u>underlining</u>	The default setting of a parameter of a compiler option or directive.	nomaf <u>maf</u>
monospace	Programming keywords and library functions, compiler builtins, examples of program code, command strings, or user-defined names.	To compile and optimize myprogram.f, enter: xlf myprogram.f -03.
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 marked bracket separators to delineate large blocks of text and icons to delineate small segments of text as follows:

Table 2. Qualifying elements

Bracket separator text	Icon	Meaning
Fortran 2003 begins / ends	 	The text describes an IBM XL Fortran implementation of the Fortran 2003 standard.
Fortran 95 begins / ends	 	The text describes an IBM XL Fortran implementation of the Fortran 95 standard.
IBM extension begins / ends	 	The text describes a feature that is an IBM XL Fortran extension to the standard language specifications.

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 \blacktriangleright — symbol indicates that a command, directive, or statement is continued from the previous line.

The — \blacktriangleleft 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 \mid — symbol and end with the — \mid symbol.

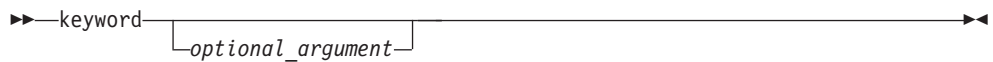
IBM XL Fortran extensions are marked by a number in the syntax diagram with an explanatory note immediately following the diagram.

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):

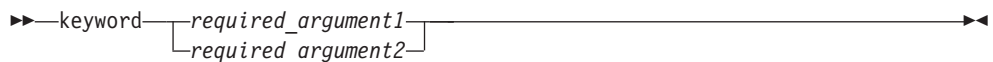


- Optional items are shown below the main path:



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.



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



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



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

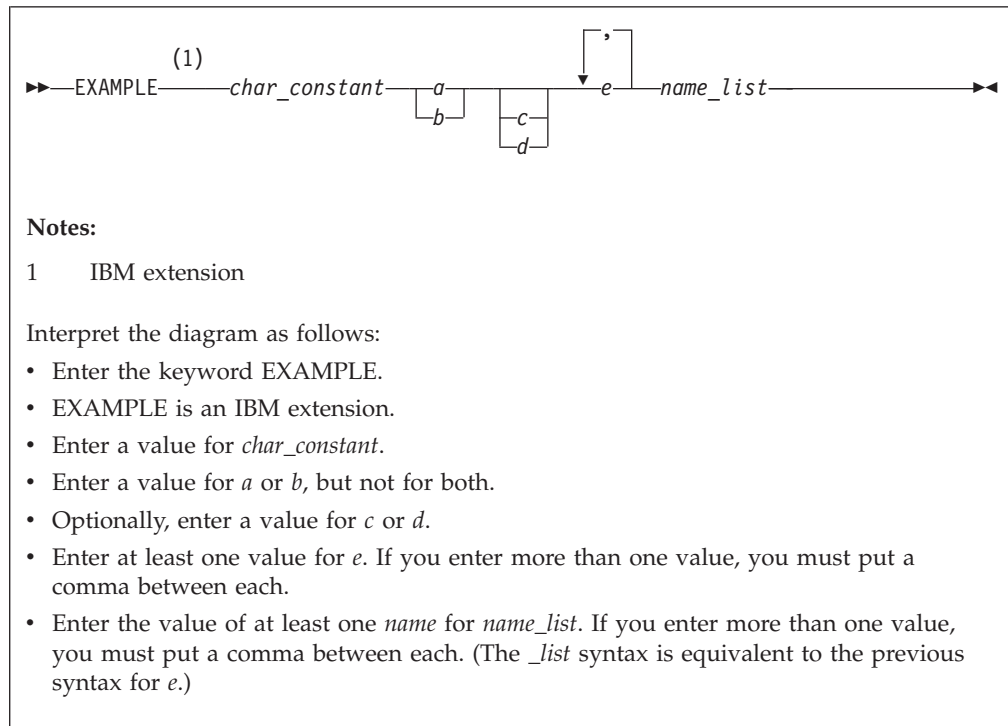


- Keywords are shown in nonitalic letters and should be entered exactly as shown.

- Variables are shown in italicized lowercase letters. They represent user-supplied names or values. If 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:



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 AIX, V13.1 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 AIX, V13.1 Installation Guide*.
 The information center is viewable on the Web at <http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp>.
- PDF documents

PDF documents are located by default in the /usr/lpp/xlf/doc/LANG/pdf/ directory, where LANG is one of en_US or ja_JP. The PDF files are also available on the Web at <http://www.ibm.com/software/awdtools/fortran/xlfortran/aix/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
<i>IBM XL Fortran for AIX, V13.1 Installation Guide, GC27-2485-00</i>	install.pdf	Contains information for installing XL Fortran and configuring your environment for basic compilation and program execution.
<i>Getting Started with IBM XL Fortran for AIX, V13.1, GI11-9418-00</i>	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.
<i>IBM XL Fortran for AIX, V13.1 Compiler Reference, SC27-2484-00</i>	compiler.pdf	Contains information about the various compiler options and environment variables.
<i>IBM XL Fortran for AIX, V13.1 Language Reference, SC27-2486-00</i>	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.
<i>IBM XL Fortran for AIX, V13.1 Optimization and Programming Guide, SC27-2487-00</i>	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 Web site 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/aix/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.)
- *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 3.0*, available at <http://www.openmp.org>

Other IBM information

- *Parallel Environment for AIX: Operation and Use*
- The IBM Systems Information Center, at <http://publib.boulder.ibm.com/infocenter/systems/index.jsp?topic=/com.ibm.aix.doc/doc/base/aixparent.htm> is a resource for AIX information.
You can find the following books for your specific AIX system:
 - *AIX Commands Reference, Volumes 1 - 6*
 - *Technical Reference: Base Operating System and Extensions, Volumes 1 & 2*
 - *AIX National Language Support Guide and Reference*
 - *AIX General Programming Concepts: Writing and Debugging Programs*
 - *AIX Assembler Language Reference*
- *ESSL for AIX V4.4 - ESSL for Linux on POWER V4.4 Guide and Reference* available at the Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL Web page.

Technical support

Additional technical support is available from the XL Fortran Support page at <http://www.ibm.com/software/awdtools/fortran/xlfortran/aix/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 Fortran, visit the product information site at <http://www.ibm.com/software/awdtools/fortran/xlfortran/aix/>.

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 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 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 AIX, V13.1 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.

Commonality with other IBM compilers

IBM XL Fortran for AIX, V13.1 is part of a larger family of IBM C, C++, and Fortran compilers.

XL Fortran, together with XL C and 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[®]/L[™], IBM Blue Gene[®]/P[™], the Cell Broadband Engine[™] architecture, 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.

Hardware and operating system support

IBM XL Fortran for AIX, V13.1 supports AIX 5L[™] for POWER[®] Version 5.3 and AIX Version 6.1. See the README file and "Before installing XL Fortran" in the *XL Fortran 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 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, `xlf`, `xlf90`, `xlf95`, and `xlf2003`. 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, `xlf_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 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 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 29.

Utilization tracking configuration file

The utilization tracking and reporting feature of the compiler has its own configuration file. The main compiler configuration file contains an entry that points to this file. The different installations of the compiler product can use a single utilization tracking configuration file to centrally manage the functionality of the utilization tracking and reporting feature. This utilization and reporting tool can be used to detect whether your organization's use of the compiler exceeds your license entitlements. For detailed information about the utilization tracking and reporting feature, see "Tracking and reporting compiler usage" in the *XL Fortran Compiler Reference*.

Language standards compliance

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)
- Extensions to the Fortran standard:
 - 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
- ISO/IEC 1539-1:2004 (referred to as Fortran 2003 or F2003)
- Common Fortran language extensions defined by other compiler vendors, in addition to those defined by IBM

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

- OpenMP V3.0 to support portable parallelized programming
- Language extensions to support vector programming

In addition to the standardized language levels, XL Fortran supports many industry language extensions, including extensions to support vector programming.

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`, or `.f03`, then use the generic compiler invocations such as `xlf` or `xlf_r` to automatically select the appropriate language level appropriate to the filename extension.

You can rebuild your FORTRAN 77, Fortran 90, Fortran 95, and Fortran 2003 source code with IBM XL Fortran for AIX, V13.1 and link them all into the same application. Similarly, object code or libraries compiled with previous versions of XL Fortran are still compatible with the newest XL Fortran compiler and runtime environment, except for two cases. See Backward compatibility issues in the *XL Fortran Compiler Reference* for more information.

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

Tools and utilities

There are many tools and utilities that are included with XL Fortran.

xlfndi This is a script you can use to install XL Fortran to a non-default directory location.

CreateExportList command

Creates a file that contains a list of all the global symbols found in a given set of object files.

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 weigh 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**).

Utilization reporting tool

The utilization reporting tool generates a report describing your organization's utilization of the compiler. These reports can be used to monitor compliance with your compiler license entitlements. You can use the **urt** command to control how the report is generated. For more information about this tool, see "Tracking and reporting compiler usage" in the *XL Fortran Compiler Reference*.

Program optimization

XL Fortran provides several compiler options that can help you control the optimization or 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. 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 POWER 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*

64-bit object capability

The XL Fortran 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 Fortran 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 XL Fortran in a 64-bit environment" 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 (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 "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 AIX, V13.1, XL Fortran supports all features of the OpenMP API Version 3.0 specification. See "OpenMP 3.0" on page 8 for an overview of the changes introduced by this feature.

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 listing and XML reports 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 "Understanding XL Fortran compiler listings" in the *XL Fortran Compiler Reference*.

It is also possible to get information from the compiler in XML 1.0 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 improve programmer productivity when tuning applications, especially high performance applications. The report is defined by an XML schema and 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. That information can be examined by **dbx** 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 Fortran for AIX, V13.1

This section describes features and enhancements added to the compiler in IBM XL Fortran for AIX, V13.1.

Support for POWER7 processors

IBM XL Fortran for AIX, V13.1 supports POWER7™ processors. New features are introduced in support of POWER7 processors.

The new features and enhancements for POWER7 processors fall into four categories:

- vector scalar extension data types and intrinsic procedures
- MASS libraries for POWER7 processors
- built-in procedures for POWER7 processors
- compiler options for POWER7 processors

Vector scalar extension data types and intrinsic procedures

This release of the compiler supports the Vector Scalar eXtension (VSX) instruction set in the POWER7 processors. New data types and intrinsic procedures are introduced to support the VSX instructions. With the VSX intrinsic procedures and the original Vector Multimedia eXtension (VMX) intrinsic procedures, you can efficiently manipulate vector operations in your application.

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

Mathematical Acceleration Subsystem (MASS) libraries for POWER7 processors

Vector libraries

The vector MASS library **libmassvp7.a** contains vector procedures that have been tuned for the POWER7 architecture. The procedures can be used in either 32-bit mode or 64-bit mode.

Procedures supporting previous POWER processors, either single-precision or double-precision, are included for POWER7 processors.

The following new procedures are added, in both single-precision and double-precision function groups:

- exp2
- exp2m1
- log21p
- log2

For more information about the vector libraries, see Using the vector libraries in the *XL Fortran Optimization and Programming Guide*.

SIMD libraries

The MASS SIMD library `libmass_simdp7.a` contains an accelerated set of frequently used math intrinsic procedures that provide improved performance over the corresponding standard system library procedures.

For more information about the SIMD libraries, see *Using the SIMD library for POWER7 in the XL Fortran Optimization and Programming Guide*.

POWER7 hardware directives and intrinsics

New hardware directives and intrinsics are added to support the following POWER7 processor features:

- new POWER7 prefetch extensions and cache control
- new POWER7 hardware instructions

For more information, see “Directives and intrinsics new for this release” on page 16.

New compiler option for POWER7 processors

New arch and tune compiler options

The `-qarch` compiler option specifies the processor architecture for which code is generated. The `-qtune` compiler option tunes instruction selection, scheduling, and other architecture-dependent performance enhancements to run best on a specific hardware architecture.

`-qarch=pwr7` produces object code containing instructions that will run on the POWER7 hardware platforms. With `-qtune=pwr7`, optimizations are tuned for the POWER7 hardware platforms.

For more information, see `-qarch` in the and `-qtune` in the *XL Fortran Language Reference*.

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:

- Support for parameterized derived types, including kind and length parameters.
- Support for generic interfaces with the same name as derived types.

OpenMP 3.0

In this release, XL Fortran complies with the OpenMP API Version 3.0 specification. The XL Fortran implementation is based on IBM's interpretation of the OpenMP Application Program Interface Version 3.0.

Features implemented for OpenMP in this release are:

- Full support for OpenMP task level parallelizations. 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.
- Allocatable arrays. You can specify allocatable arrays on `PRIVATE`, `FIRSTPRIVATE`, `LASTPRIVATE`, `REDUCTION`, `COPYIN`, and `COPYPRIVATE` clauses.

- Nested parallelism. A list of runtime routines are available for you to set or get the nested levels and thread limit. These include:
 - `omp_get_thread_limit`
 - `omp_get_max_active_levels`
 - `omp_set_max_active_levels`

To return nested parallelism information, you can now use the following routines:

- `omp_get_level`
- `omp_get_ancestor_thread_num`
- `omp_get_team_size`
- `omp_get_active_level`
- Stack size control. You can now control the size of the stack for threads created by the OpenMP runtime library using the new environment variable `OMP_STACKSIZE`.
- Users can give hints to the desired behavior of waiting threads using the new environment variable `OMP_WAIT_POLICY`.
- The storage of a private variable is not allowed to reuse the storage of the original variable on the master thread.
- Some restrictions on the `PRIVATE` clause have been removed. A variable that appears in the `REDUCTION` clause of a parallel construct can now also appear in a `PRIVATE` clause on a work-sharing construct.
- A new `SCHEDULE` type, `AUTO`, allows the compiler and runtime system to control scheduling. You can use the following routines to get or set schedule type:
 - `omp_get_schedule`
 - `omp_set_schedule`
- Consecutive loop constructs with `STATIC` schedule with `NOWAIT` clause now guarantee the same iterations are being assigned to the same thread in the constructs.
- You can set the `OMP_THREAD_LIMIT` environment variable to determine the number of OpenMP threads to use for the whole program. Set `OMP_MAX_ACTIVE_LEVELS` if you want to control the maximum number of nested, active parallel regions.

The following is an enhancement to OpenMP in this release:

- You can specify the `-qsmp=ostls` option to use Thread Local Storage (TLS) provided at operating system (OS) level to implement `THREADPRIVATE` data. However, your operating system must already support TLS for you to use the `-qsmp=ostls` suboption. Use `-qsmp=noostls` to disable OS level TLS support.

For more information, see:

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

Performance and optimization

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

Enhancements to -qpdf

The use of the **-qpdf** option consists of two steps. First, compile your program with **-qpdf1** and run it with a typical set of data to generate the profiling data. Second, compile your program again with **-qpdf2** to optimize the program based on the profiling data.

In previous releases, if you modified the source file and compiled with the **-qpdf** option, the compilation would stop with an error. As of IBM XL Fortran for AIX, V13.1, you can use profiling data after you modify your source files. To do this, compile your application using the stale profiling data at the second stage of the PDF process.

Three new suboptions are added to the **-qpdf** option. These new suboptions allow more fine-grained control over performance improvement and extend **-qpdf** to support multiple pass profiling, cache miss profiling, block counter profiling, call counter profiling, and extended value profiling.

The three new **-qpdf** suboptions are:

level Supports multiple-pass profiling, cache miss profiling, block counter profiling, call counter profiling, and extended value profiling. You can compile your application with **-qpdf1=level=0|1|2** to generate profiling data with different levels of optimization.

Note: Both **-qpdf1=level=0** and **-qpdf1=level=1** support single-pass profiling, whereas **-qpdf1=level=2** supports multiple-pass profiling.

exename

Generates the name of the PDF file you specify with the **-o** parameter.

defname

Reverts the PDF file to its default file name.

For detailed information about these suboptions, see **-qpdf1**, **-qpdf2** in the *XL Fortran Compiler Reference*.

Reports about compiler optimizations

There have been a number of enhancements to the listing reports to give you more information on how the compiler optimized your code. You can use this information to get further benefits from the compiler's optimization capabilities. For more details about these enhanced reports, see "New diagnostic reports" on page 11.

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, see "Optimization and tuning options" in the *XL Fortran Compiler Reference*.

Table 4. Performance-related compiler options and directives

-qhot	Two suboptions -qhot=fastmath and -qhot=nofastmath are added to -qhot , to tune your applications to use the fast scalar versions of the math routines or to use the default versions. -qhot=level=2 is also added for loop transformation analysis of nested loops. For details, see the -qhot section in the <i>XL Fortran Compiler Reference</i> .
-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. <i>number</i> is a range of integer values between 0 and 10 that indicates the level of inlining you want to use. For details, see -qinline in the <i>XL Fortran Compiler Reference</i> .
-qipa	A new enhancement added to -qipa is -r -qipa=relink . You can generate relinkable objects while preserving IPA information by specifying -r -qipa=relink . This creates a nonexecutable package that contains all object files. By using this suboption, you can postpone linking until the very last stage. -qipa=clonearch is no longer supported. Consider using -qtune=balanced . For detailed information, see -qipa section in the <i>XL Fortran Compiler Reference</i> .
-qpdf	-qpdf provides suboptions to give you more control flexibility in controlling different PDF optimizations. For more information, see the -qpdf1 , -qpdf2 section in the <i>XL Fortran Compiler Reference</i> .
-qprefetch	A new enhancement added to -qprefetch is -qprefetch=assistthread . -qprefetch inserts prefetch instructions automatically where there are opportunities to improve code performance. For details, see -qprefetch in the <i>XL Fortran Compiler Reference</i> .

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 to identify opportunities to improve performance of code.

Compiler reports in XML format

It is now possible to get information in XML 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 improve programmer productivity when tuning applications, especially high performance applications.

The information from the compiler is produced in XML 1.0 format. The report is defined by an XML schema and is easily consumable by tools that you can create to read and analyze the results. A stylesheet, `x1style.xml`, is provided to render the report into a human readable format that can be read by anyone with a browser which supports XSLT.

In this release, there are four optimization categories in the report, one for inlining, one for loop transformations, one for data reorganizations, and one for profile-directed feedback information. The new **-qlistfmt** option and its associated suboptions generate the new XML 1.0 report.

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*.

Enhancements to profiling reports

Additional sections of the listing report have been added to help you understand your programs. When using the **-qreport** option with the **-qpdf2** option, you can get the following additional sections added to your listing file in the section entitled PDF Report:

Loop iteration count

The most frequent loop iteration count and the average iteration count, for a given set of input data, is calculated for most loops in a program. This information is only available when the program is compiled at optimization level **-O5**.

Block and call count

This section of the report covers the call structure of the program and the respective execution count for each called function. It also includes block information for each function. For non-user defined functions, only execution count is given. The total block and call coverage, and a list of the user functions ordered by decreasing execution count are printed in the end of this report section. In addition, the block count information is printed at the beginning of each block of the pseudo-code in the listing files.

Cache miss

This section of the report is printed in a single table. It reports the number of cache misses for certain functions, with additional information about the functions such as: cache level, cache miss ratio, line number, file name, and memory reference.

Note: You must use the **-qpdf1=level=2** option to get this report. You can also select the level of cache to profile using the **PDF_PM_EVENT** environment variable during run time.

For detailed information about the profile-directed feedback, see "Using profile-directed feedback" in the *XL Fortran Optimization and Programming Guide*.

For additional information about the listing files, see "Understanding XL Fortran compiler listings" in the *XL Fortran Compiler Reference*

Report of data reorganization

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

- Data reorganizations (a summary of useful information about 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 some 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.

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	Generates a report in an XML 1.0 format containing information about optimizations performed by the compiler and missed optimization opportunities. The report contains information about inlining, loop transformations, data reorganization and profile-directed feedback.
-qreport	The listing now contains a PDF report section when used with -ppdf2 . Another new section in the listing files is a DATA REORGANIZATION section when used with -qipa=level=2 or -O5 .

Utilization tracking and reporting

The utilization tracking and reporting feature is a lightweight and simple mechanism for tracking the compiler utilization within your organization. It is disabled by default. You can use this feature to detect whether your organization's use of the compiler exceeds your compiler license entitlements.

When utilization tracking is enabled, each invocation of the compiler is recorded in a compiler utilization file. You can run the utilization reporting tool to generate a

report from one or more of these files to get a picture of the overall usage of the compiler within your organization. The `urt` command can be used to control how the report is generated. In particular, the report indicates whether the compiler usage complies with the number of Concurrent User licenses that you have purchased.

The utilization tracking and reporting feature is easy to set up and manage, and utilization tracking does not impact the usage or performance of the compiler.

For detailed information about the utilization tracking and reporting feature, see "Tracking and reporting compiler usage" in the *XL Fortran Compiler Reference*.

New or changed compiler options and directives

New and changed compiler options and directives are described in this section.

Compiler options can be specified on the command line or 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.

Table 6. New or changed compiler options and directives

Option/directive	Description
<code>-qarch</code>	A new suboption has been added to <code>-qarch</code> , specifying <code>-qarch=pwr7</code> produces object code that contains instructions that run on the POWER7 hardware platforms.
<code>-qassert</code>	New suboptions have been added for <code>-qassert</code> to provide more information about the characteristics of the files that can help to fine-tune optimizations. The <code>-qassert=contig</code> 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 <code>-qassert=refalign</code> option tells the compiler that all pointers only point to naturally-aligned data.
<code>-qbindcextname</code>	You can use the <code>-qbindcextname</code> option to control whether the <code>-qextname</code> option affects <code>BIND(C)</code> entities.
<code>-qfunctrace</code>	Inserts calls to user-defined tracing procedures at procedure entry and exit.
<code>-qfunctrace_xlf_catch</code>	You can use this option to specify the name of the catch tracing subroutine.
<code>-qfunctrace_xlf_enter</code>	You can use this option to specify the name of the entry tracing subroutine.
<code>-qfunctrace_xlf_exit</code>	You can use this option to specify the name of the exit tracing subroutine.

Table 6. New or changed compiler options and directives (continued)

Option/directive	Description
-qhot	<p>A new suboption has been added for -qhot. The -qhot compiler option is a powerful alternative to hand tuning that provides opportunities to optimize loops and array language.</p> <p>The -qhot=fastmath option enables the replacement of math routines with available math routines from the XLOPT library only if -qstrict=nolibrary is enabled. -qhot=nofastmath disables the replacement of math routines by the XLOPT library. -qhot=fastmath is enabled by default if -qhot is specified regardless of the hot level.</p>
-qinline	Attempts to inline functions instead of generating calls to those functions, for improved performance.
-qlibmpi	You can use -qlibmpi to tune code based on the known behavior of the Message Passing Interface (MPI) functions.
-qlistfmt	Generates a report in an XML 1.0 format containing information about some optimizations performed by the compiler and some missed optimization opportunities for inlining, loop transformations, data reorganization and profile-directed feedback.
-qmkshrobj	You can use -qmkshrobj to create a shared object from generated object files.
-qpdf1,-qpdf2	New options have been added to -qpdf1,-qpdf2 .
-qprefetch	A new suboption has been added to -qprefetch . When you work with applications that generate a high cache-miss rate, you can use -qprefetch=assistthread to exploit assist threads for data prefetching.
-qsaveopt -qnosaveopt	The existing -qsaveopt option is enhanced to also include the user's configuration file name and the options specified in the configuration files.
-qsimd	Controls whether the compiler can automatically take advantage of vector instructions for processors that support them.
-qstackprotect	You can use -qstackprotect to protect your applications against malicious code or programming errors that overwrite or corrupt the stack.
-qstrict	<p>A new suboption has been added to the -qstrict option to allow more control over optimizations and transformations that violate strict program semantics.</p> <p>-qstrict=vectorprecision disables vectorization in loops where it might produce different results in vectorized iterations than in nonvectorized ones.</p>
-qtune	A new suboption has been added to -qtune . If you specify -qtune=pwr7 , optimizations are tuned for the POWER7 hardware platforms.

Table 7. Deprecated directives and options

Option/directive	Description
SCHEDULE	This directive is deprecated and might be removed in a future release. You can use the OpenMP equivalent.
-Q	This option is deprecated and replaced with -qinline .
-qenablevmx	This option is deprecated and replaced with the -qsimd=auto option.
-qhot=simd nosimd	-qhot=simd nosimd are deprecated and might be removed in a future release. You can use -qsimd .
-qinfo=private	-qinfo=private is deprecated and replaced with -qreport .
-qinfo=reduction	-qinfo=reduction is deprecated and replaced with -qreport .
-qipa=inline noinline	-qipa=inline noinline are deprecated and might be removed in a future release. You can use -qinline .
-qipa=clonearch noclonearch	-qipa=clonearch noclonearch is no longer supported. You can use -qtune=balanced .
-qipa=clonearch noclonearch	-qipa=cloneproc nocloneproc is no longer supported. You can use -qtune=balanced .

Directives and intrinsics new for this release

This section lists directives and intrinsics that are new for this release.

VSX built-in functions

Vector Scalar eXtension (VSX) is newly added for POWER7 processors.

For more information about VSX built-in functions, see Vector intrinsic procedures.

POWER7 prefetch extensions and cache control

The POWER7 processor has cache control and stream prefetch extensions that support store stream prefetch and prefetch depth control. XL Fortran provides the following new directives to provide direct programmer access to these instructions:

- PROTECTED_STREAM_STRIDE
- TRANSIENT_PROTECTED_STREAM_COUNT_DEPTH
- UNLIMITED_PROTECTED_STREAM_DEPTH
- TRANSIENT_UNLIMITED_PROTECTED_STREAM_DEPTH
- PARTIAL_DCBT
- DCBTT
- DCBTSTT
- DCBFLP

The compiler can insert the directives automatically when it optimizes the code. You can disable automatic use of these instructions with **-qnoprefetch**.

For more information about the directives, see Hardware-specific directives in the *XL Fortran Language Reference*.

POWER7 intrinsics

New XL Fortran built-in functions corresponding to each new POWER7 hardware instruction are added in this release. With these functions, you can directly manipulate specific hardware instructions in your code, which can improve the performance of your application.

- BPERMD
- DIVDE
- DIVWE

Comparison intrinsics

This new function compares bytes.

- CMPB

Compatibility of redistributable library libxlopt.a

Starting from this release, compatibility of the redistributable library, libxlopt.a, will be maintained. The libxlopt.a library will be compatible with compiler XL Fortran for AIX, V13.1 or later.

Previously, the version of the redistributable library had to be the same as the version of the compiler with which the application was compiled.

You can download and use the latest redistributable library for multiple applications compiled with XL Fortran V13.1, or later, on supported platforms.

Chapter 3. Enhancements added in previous versions

This section lists the enhancements added to the compiler in the previous releases.

Enhancements added in Version 12.1

This section describes features and enhancements added to the compiler in Version 12.1.

Operating system support

IBM XL Fortran for AIX, V12.1 supports AIX V6.1 and AIX V5.3.

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

XL Fortran language-related updates

Fortran 2003 enhancements

The OPEN and INQUIRE statements have been updated with the ENCODING= specifier to indicate the encoding form of the file.

IEEE module enhancements

The IEEE_ARITHMETIC module defines a new constant, IEEE_OTHER_VALUE.

The IEEE_ARITHMETIC module defines three new functions:
IEEE_SET_UNDERFLOW_MODE, IEEE_GET_UNDERFLOW_MODE and
IEEE_SUPPORT_UNDERFLOW_MODE.

OpenMP 3.0

In IBM XL Fortran for AIX, V12.1, XL Fortran has added some of the features of the OpenMP API Version 3.0 specification. The XL Fortran 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 implemented in this release 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.
- Nesting support - a COLLAPSE clause has been added to the DO, and PARALLEL DO directives to allow parallelization of perfect loop nests. This means that multiple loops in a nest can be parallelized.

For more information, see:

- "Parallel programming with XL Fortran" in the *XL Fortran 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 `-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. Here is a list of suboption groups:

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 individual operations can be reordered in a way that violate program language semantics.

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

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

For detailed information about these suboptions, see "`-qstrict`" in the *XL Fortran 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, see "Optimization and tuning options" in the *XL Fortran Compiler Reference*.

Table 8. Performance-related compiler options and directives

Option/directive	Description
<code>EXECUTION_FREQUENCY</code>	The <code>EXECUTION_FREQUENCY</code> 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.

Table 8. Performance-related compiler options and directives (continued)

Option/directive	Description
-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	When -qsmp=omp is in effect, some of the additional functionality of OpenMP API 3.0 is now available. For more information, see “OpenMP 3.0” on page 8.

For additional information about performance tuning and program optimization, see "Optimizing your applications" in the *XL Fortran 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 Fortran Compiler Reference* for detailed descriptions and usage information for these and other compiler options.

Table 9. 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 9 for more information.
-qfpp	Allows Fortran-specific preprocessing features in the C preprocessor which ships with XL Fortran. Since it is a C preprocessor option it is invoked with the -WF option as -WF, -qfpp .
-qppsuborigarg	Instructs the C preprocessor to substitute original macro arguments before further macro expansion. Since it is a C preprocessor option it is invoked with the -WF option as -WF, -qppsuborigarg .
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.
-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	When -qsmp=omp is in effect, some of the additional functionality of OpenMP API 3.0 is now available. For more information, see “OpenMP 3.0” on page 19.
-qtimestamps	This option can be used to remove timestamps from generated binaries.

Enhancements added in Version 11.1

This section describes features and enhancements added to the compiler in Version 11.1.

Enhanced support for Fortran 2003

IBM XL Fortran for AIX, V11.1 provides one of the most complete Fortran 2003 implementations currently available.

The enhanced compliance with the Fortran 2003 standard includes:

- Implementation of the full Fortran 2003 object-oriented programming model, including:
 - Type extension
 - Type-bound procedures
 - Type finalization
 - Polymorphism and runtime type determination including the SELECT TYPE construct
 - Abstract and generic interfaces
 - Declaration of abstract types and deferred bindings
 - PASS attribute
- I/O enhancements
 - User-defined derived type I/O
 - New I/O specifiers including SIGN= and DECIMAL= (DC and DP edit descriptors)
 - Asynchronous I/O as defined by Fortran 2003 including the WAIT statement
 - User-specified control of rounding during format conversion using the ROUND= specifier and new edit descriptors
 - Handling of IEEE infinity and not-a-number in REAL and COMPLEX editing
 - Support for the PAD= specifier on INQUIRE operations
 - Support for the PAD= and BLANK= specifiers on READ statements
 - Support for the DELIM= specifier on WRITE statements
 - Support for the stream access method
- Scoping and data manipulation enhancements
 - Renaming of defined operators on USE statements
 - Fortran 2003 VOLATILE statement
 - COMPLEX literals
 - Pointer assignment and initialization expression enhancements
 - Improved structure constructors
 - Allocatable objects beyond Fortran 95 arrays
 - Allocatable enhancements including resizing on assignment and MOVE_ALLOC intrinsic
 - Explicit type specification in an array constructor
 - ASSOCIATE construct
 - Support for enumerators
 - BIND(C) for portable interoperability with C code
- Procedure enhancements
 - Generic bindings for interfaces, defined operators, and defined assignment

- VALUE attribute for characters of length greater than one and derived types with allocatable components
- Procedure pointers, procedure declaration statement, and procedure pointers as derived type components
- Generalization of the MODULE PROCEDURE statement
- Deferred CHARACTER length
- Intrinsic function enhancements
 - IEEE and Fortran environment modules
 - MAX, MIN, MAXLOC, MINLOC, MAXVAL, and MINVAL intrinsics for character types
 - Allow REAL type for COUNT_RATE argument of SYSTEM_CLOCK
 - Allow boz-literal constants on INT, REAL, CMPLX, and DBL intrinsics
 - Allow a new KIND argument on all intrinsics mandated by Fortran 2003
 - Returning signed zero results from the ATAN2, LOG, and SQRT intrinsics
 - Added SELECTED_CHAR_KIND intrinsic
- Other enhancements
 - Enhanced STOP statement
 - Increased the maximum number of continuation lines

Fortran 2003 compiler invocations and file types

Compiler invocation commands instruct the compiler to adhere more closely to Fortran 2003 language standards when compiling your applications. The new invocations are:

- xlf2003
- xlf2003_r (for threaded applications)
- f2003

These invocations provide partial compliance to the Fortran 2003 standard. You can obtain behavior that complies to the Fortran 2003 standard by doing the following:

1. Set the XLFRTEOPTS environment variable to `"err_recovery=no:langlvl=2003std:iostat_end=2003std:internal_nldelim=2003std"`
2. Invoke the compiler with the following option settings: `"-qlanglvl=2003std -qnodirective -qnoescape -qextname -qfloat=nomaf:rndsngl:nofold -qnoswapomp -qstrictieemod"`

In addition to the compiler invocations, this XL Fortran also supports new filename extensions:

- .f03
- .F03 (invokes the C preprocessor before compiling)

Compiler option -qxlf2003

XL Fortran supports a compatibility option, **-qxlf2003**. This option provides backward compatibility with XL Fortran V10.1 and the Fortran 2003 standard for certain aspects of the language.

When compiling with the Fortran 2003 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 **-qxf2003=nopolymorphic**.

The **-qxf2003** compiler option also includes several other suboptions to provide backward compatibility with earlier versions of XL Fortran. See **-qxf2003** in the *XL Fortran Compiler Reference* for more information.

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 arithmetic computations with REAL(4) data types 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 Fortran Version 11.1 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**

Support removed for selected processors

XL Fortran Version 11.1 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 | 603**
- **-qtune= pwr | pwr2 | pwr2s | pwrx | p2sc | 601 | 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 Fortran Compiler Reference*.

Table 10. Performance-related compiler options and directives

Option/directive	Description
-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.
-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=normdsngl to obtain the same computational results.
-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.

Table 10. Performance-related compiler options and directives (continued)

Option/directive	Description
-qpdf	The -qpdf option can now be used to provide profile-directed feedback on specific objects. See "Object level profile-directed feedback" in the <i>XL Fortran Optimization and Programming Guide</i> 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.
EXPECTED_VALUE(param, value)	Use the 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.

Directives and intrinsics in Version 11.1

Some directives and intrinsics were added in Version 11.1.

PowerPC cache control

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

- **DCBST(variable) ! Data Cache Block Store**
- **DCBF(variable) ! Data Cache Block Flush**

For more information, see "Hardware-specific directives" in the *XL Fortran Language Reference*.

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 Fortran provides the following new directives to provide direct programmer access to these instructions.

- **DCBFL(variable); ! pwr6 - Data Cache Block Flush from L1 data cache only**
- **PROTECTED_UNLIMITED_STREAM_SET_FORWARD(prefetch_variable, stream_id) ! pwr5 and pwr6**
- **PROTECTED_UNLIMITED_STREAM_SET_BACKWARD(prefetch_variable, stream_id) ! pwr5 and pwr6**
- **PROTECTED_UNLIMITED_STORE_STREAM_SET_FORWARD(prefetch_variable, stream_id) ! pwr6**
- **PROTECTED_UNLIMITED_STORE_STREAM_SET_BACKWARD(prefetch_variable, stream_id) ! pwr6**
- **PROTECTED_STORE_STREAM_SET_FORWARD(prefetch_variable, stream_id) ! pwr6**
- **PROTECTED_STORE_STREAM_SET_BACKWARD(prefetch_variable, stream_id) ! pwr6**
- **PROTECTED_STREAM_COUNT_DEPTH(unit_count, prefetch_depth, stream_id) ! 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 Fortran Compiler Reference* for detailed descriptions and usage information for these and other compiler options.

Table 11. Other new or changed compiler options

Option/directive	Description
-qalias_size=bytes	The -qalias_size option helps you avoid memory fragmentation in aliasing tables by letting you specify an appropriate initial size for those tables. This option can also be specified in your program source with @PROCESS ALIAS_SIZE(bytes) .
-qdescriptor= v1 v2	The -qdescriptor option lets you select the XL Fortran internal descriptor data structure format used in your compiled applications. The default v1 suboption uses a data structure format that provides backwards compatibility with objects compiled with XL Fortran V10.1 and earlier. Regardless of what -qdescriptor setting is in effect, applications containing object-oriented constructs will use the v2 suboption data structure format for those constructs, and will not be compatible with objects compiled with XL Fortran V10.1 and earlier. You should consider explicitly specifying the v2 suboption if you are compiling new applications that will not need to interact with objects compiled with XL Fortran V10.1 and earlier.
-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 11.1, 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 stackcheck option of the XLSMPOPTS environment variable.
-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 4. 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*.

Upgrading to XL Fortran V13.1

XL Fortran V13.1 is compatible with previous versions of XL Fortran. From time to time, new language features, language standard interpretations, and performance issues might force changes or option defaults, or raise other compatibility issues.

Please refer to Upgrading to XL Fortran V13.1 in the *XL Fortran Language Reference* for more details.

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*.

Configuring compiler utilization tracking and reporting

In addition to the compiler configuration file, there is a separate configuration file for the utilization tracking and reporting feature. Utilization tracking is disabled by default, but you can enable it by modifying an entry in this configuration file. Various other aspects of utilization tracking can also be configured using this file.

Although the compiler configuration file is separate from the utilization tracking configuration file, it contains an entry that specifies the location of the utilization tracking configuration file so that the compiler can find this file.

For more information about how to configure the utilization tracking and reporting feature, see Tracking and reporting compiler usage in the *XL Fortran Compiler Reference*.

Chapter 5. 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 is 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, such as `vi` or `emacs`.

Source programs must be saved using a recognized file name suffix. See the “XL Fortran input and output files” on page 35 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 77 programs

Where possible, using the `xlf` compiler invocation maintains compatibility with existing programs by using the same I/O formats as FORTRAN 77 and some implementation behaviors compatible with earlier versions of XL Fortran.

The `f77` compiler invocation is identical to `xlf`, assuming that you have not customized the configuration file.

Though you may need to continue using these invocations for compatibility with existing makefiles and build environments, programs compiled with these invocations may not conform to the Fortran 2003, Fortran 95, or Fortran 90 language level standards.

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
`f95, xlf95`

Fortran 90
`f90, xlf90`

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:rndsngl:nofold -qnoswapomp`

For full Fortran 95 compliance:
`-qlanglvl=95std -qnodirective -qnoescape -qextname
-qfloat=nomaf:rndsngl: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 XLF RTEOPTS="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**.

Compiling Fortran 2003 programs

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

Fortran 2003

```
f2003, xlf2003
```

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:rndsngl:nofold -qnoswapomp -qstrictieemod
```

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

```
export XLF RTEOPTS="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 backward compatibility with XL Fortran V10.1 and the Fortran 2003 standard for certain aspects of the language.

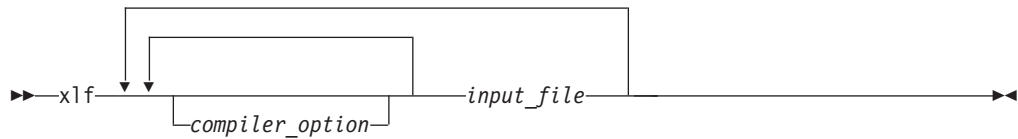
When compiling with the Fortran 2003 compiler invocations, the default setting is **-qxlf2003=polymorphic**. This setting instructs the compiler to allow polymorphic items such as the CLASS type specifier and SELECT TYPE construct in your Fortran application source.

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

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 the basic invocation syntax shown below:



For most applications, you should compile with `xlf` 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 `.f03`, `.f95`, `.f90`, or `.f77`, compiling with `xlf`, 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:

- `xlf_r`, `xlf_r7`

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

POSIX Pthreads API support

On AIX Version 5.1 and higher, XL Fortran supports 64-bit thread programming with the 1003.1-1996 (POSIX) standard Pthreads API. It also supports 32-bit programming with both the Draft 7 and the 1003.1-1996 standard APIs.

You can use invocation commands (which use corresponding stanzas in the `xlf.cfg` configuration file) to compile and then link your programs with either the 1003.1-1996 standard or the Draft 7 interface libraries.

- To compile and then link your program with the 1003.1-1996 standard interface libraries, use the `_r` variants of the compiler invocation commands. For example, you could specify:

```
fortran_r test.f
```

- To compile and then link your program with the Draft 7 interface libraries, use the `_r` variants of the compiler invocation commands. For example, you could specify:

```
fortran_r7 test.f
```

Apart from the level of thread support, the `_r7` invocation variants and their corresponding stanzas in the `vac.cfgvac.cfgxlf.cfg` configuration file provide the same support as their corresponding `_r` counterparts.

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.

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 12. Input file types

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

Table 13. 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 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:

```
xlf 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.

```
xlf -c file1.f           # Produce one object file (file1.o)
xlf -c file2.f file3.f  # Or multiple object files (file2.o, file3.o)
xlf 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*.

Linking new objects with existing ones

If you have `.o` or other object files that you compiled with an earlier version of XL Fortran, you can link them with object files that you compile with the current level of XL Fortran.

See "Linking new objects with existing ones" in the *XL Fortran Compiler Reference* for more information.

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:

```
xlf95 -omansion front_door.f entry_hall.f parlor.f sitting_room.f \  
      master_bath.f kitchen.f dining_room.f pantry.f utility_room.f  
  
vi kitchen.f # Fix problem in OVEN subroutine  
xlf95 -o newmansion kitchen.f 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 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.

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 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.

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, 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 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*.

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 Fortran 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 Fortran Runtime Environment PTF images, together with licensing and usage information, from the XL Fortran Support page at:

www.ibm.com/software/awdtools/fortran/xlfortran/support

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.

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

You can then use **dbx** 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 Fortran Optimization and Programming Guide*.

Determining what level of XL Fortran is installed

If contacting software support for assistance, you will need to know what level of XL Fortran 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:

```
xlf -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, 2010. All rights reserved.

This software and documentation are based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California. We acknowledge the following institution for its role in this product's development: the Electrical Engineering and Computer Sciences Department at the Berkeley campus.

Trademarks and service marks

IBM, the IBM logo, and [ibm.com](http://www.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

- .a files 36
- .f and .F files 36
- .i files 36
- .lst files 36
- .mod files 36
- .o files 36
- .s files 36
- .S files 36

Numerics

- 64-bit environment 5

A

- a.out file 36
- archive files 36
- assembler
 - source (.s) files 36
 - source (.S) files 36

B

- basic example, described ix

C

- code optimization 4
- compilation
 - sequence of activities 31
- compiler
 - controlling behavior of 35
 - invoking 32
 - running 32
- compiler directives
 - new or changed 14
- compiler options
 - conflicts and incompatibilities 35
 - new or changed 14
 - specification methods 35
- compiling
 - SMP programs 34

D

- dbx debugger 6, 39
- debugger support 39
 - output listings 39
 - symbolic 6
- debugging 39
- debugging compiled applications 39
- debugging information, generating 39
- directives 16, 26
- dynamic linking 37

E

- editing source files 31
- executable files 36
- executing a program 38
- executing the linker 36

F

- f2003 command
 - level of Fortran standard compliance 33
- f77 command
 - description 32
 - level of Fortran standard compliance 32
- f90 command
 - description 32
- f95 command
 - description 32
- files
 - editing source 31
 - input 36
 - output 36
- fort77 command
 - description 32
- Fortran 2003
 - compiling programs written for 33
- Fortran 90
 - compiling programs written for 32
- Fortran 95
 - compiling programs written for 32

I

- input files 36
- invocation commands 34
- invoking a program 38
- invoking the compiler 32

L

- language standards 2
- language support 2
- level of XL Fortran, determining 39
- libraries 36
- linking
 - dynamic 37
 - static 37
- linking process 36
- listings 36

M

- migration
 - source code 35
- mod files 36
- multiprocessor systems 5, 8, 19

O

- object files 36
 - creating 36
 - linking 36
- OMP directives 8, 19
- OpenMP 5
- optimization
 - programs 4
- output files 36

P

- parallelization 5, 8, 19
- performance
 - optimizing transformations 4
- POSIX Pthreads
 - API support 34
- problem determination 39
- programs
 - running 38

R

- running the compiler 32
- runtime
 - libraries 36
- runtime environment 38
- runtime options 38

S

- shared memory parallelization 5, 8, 19
- shared object files 36
- SMP
 - programs, compiling 34
- SMP programs 5
- source files 36
- source-level debugging support 6
- static linking 37
- symbolic debugger support 6

T

- tools 3
 - cleanpdf utility 3
 - CreateExportList 3
 - custom installation 3
 - install 3
 - mergepdf utility 4
 - resetpdf utility 4
 - showpdf utility 4
 - urt 3
 - xlfindi 3

U

- utilities 3
 - cleanpdf 3

utilities (*continued*)

- CreateExportList 3
- custom installation 3
- install 3
- mergepdf 4
- resetpdf 4
- showpdf 4
- urt 3
- xlfindi 3

V

- vac.cfg file 35

X

- xl command
 - description 32
 - level of Fortran standard compliance 32, 33
- xl_r command
 - description 32
 - for compiling SMP programs 34
 - level of Fortran standard compliance 32
- xl_r7 command
 - description 32
 - for compiling SMP programs 34
 - level of Fortran standard compliance 32, 33
- xl2003 command
 - level of Fortran standard compliance 33
- xl2003_r command
 - level of Fortran standard compliance 33
- xl2003_r7 command
 - level of Fortran standard compliance 33
- xl90 command
 - description 32
 - level of Fortran standard compliance 32
- xl90_r command
 - description 32
 - for compiling SMP programs 34
 - level of Fortran standard compliance 32
- xl90_r7 command
 - description 32
 - for compiling SMP programs 34
 - level of Fortran standard compliance 32
- xl95 command
 - description 32
- xl95_r command
 - description 32
 - for compiling SMP programs 34
 - level of Fortran standard compliance 32
- xl95_r7 command
 - description 32
 - for compiling SMP programs 34
 - level of Fortran standard compliance 32



Program Number: 5724-X15

Printed in USA

GI11-9418-00

