

IBM XL Fortran for AIX, V15.1.3



Getting Started with XL Fortran

Version 15.1.3

IBM XL Fortran for AIX, V15.1.3



Getting Started with XL Fortran

Version 15.1.3

Note

Before using this information and the product it supports, read the information in "Notices" on page 55.

First edition

This edition applies to IBM XL Fortran for AIX, V15.1.3 (Program 5765-J09; 5725-C74) 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, 2015.

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
Operating system and hardware support	1
A highly configurable compiler	1
Language standard compliance	3
Source-code migration and conformance checking	3
Tools, utilities, and commands	4
Program optimization	5
64-bit object capability	5
Shared memory parallelization	6
Diagnostic reports	7
Symbolic debugger support	7
Chapter 2. What's new for IBM XL Fortran for AIX, V15.1.3	9
Language features	9
Intrinsic procedures	10
Other enhancements	10
Chapter 3. Migration of your applications	13
Things to note in IBM XL Fortran for AIX, V15.1.3	13
Avoiding or fixing upgrade problems	14
Compatibility with earlier versions	17
Chapter 4. Enhancements added in earlier releases	21
Enhancements added in Version 15.1.2	21
Fortran 2008 features	21
Language interoperability features	21
Intrinsic procedures	22
Commands	22
Compiler options	22
Other XL Fortran updates	23
Enhancements added in Version 15.1	23

Support for POWER8 processors	23
Fortran 2008 features	25
Language interoperability features	27
OpenMP 4.0	28
Directives and intrinsic procedures	29
Compiler options	31
Other XL Fortran updates	33
Enhancements added in Version 14.1	34
Fortran 2008 features	34
OpenMP 3.1	37
Performance and optimization	38
Diagnostic reports	38
Compiler options and directives	40

Chapter 5. Setting up and customizing XL Fortran	43
Using custom compiler configuration files	43
Configuring compiler utilization tracking and reporting	43

Chapter 6. Developing applications with XL Fortran	45
The compiler phases	45
Editing Fortran source files	45
Compiling with XL Fortran	46
Invoking the compiler	46
Compiling parallelized XL Fortran applications	48
Specifying compiler options	49
XL Fortran input and output files	50
Linking your compiled applications with XL Fortran	51
Linking new objects with existing ones	51
Relinking an existing executable file	51
Dynamic and static linking	52
Running your compiled application	52
XL Fortran compiler diagnostic aids	53
Debugging compiled applications	53
Determining which level of XL Fortran is being used	54

Notices	55
Trademarks	57

Index	59
------------------------	-----------

About this document

This document contains overview and basic usage information for the IBM® XL Fortran for AIX®, V15.1.3 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, 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 about 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 behavior of the compiler. You can, however, substitute other forms of the compiler invocation command if your particular environment requires it, and compiler option usage remains the same unless otherwise specified.

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

Conventions

Typographical conventions

The following table shows the typographical conventions used in the IBM XL Fortran for AIX, V15.1.3 information.

Table 1. *Typographical conventions*

Typeface	Indicates	Example
lowercase bold	Invocation commands, executable names, and compiler options.	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. The default file name for the executable program is <code>a.out</code> .









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	Examples of program code, reference to program code, file names, path names, 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 icons and marked bracket separators to delineate the Fortran language standard text as follows:

Table 2. Qualifying elements

Icon	Bracket separator text	Meaning
 F2008  F2008	Fortran 2008 begins / Fortran 2008 ends	The text describes an IBM XL Fortran implementation of the Fortran 2008 standard.
 F2003  F2003	Fortran 2003 begins / Fortran 2003 ends	The text describes an IBM XL Fortran implementation of the Fortran 2003 standard, and it applies to all later standards.
 IBM  IBM	IBM extension begins / IBM extension ends	The text describes a feature that is an IBM XL Fortran extension to the standard language specifications.
 TS 29113  TS 29113	TS 29113 begins / TS 29113 ends	The text describes an IBM XL Fortran implementation of Technical Specification 29113, referred to as TS 29113.

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

Syntax diagrams

Throughout this information, diagrams illustrate XL Fortran syntax. This section helps 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 \blacktriangleright — symbol indicates the beginning of a command, directive, or statement.

The — \blacktriangleright 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 |— symbol and end with the —| symbol.

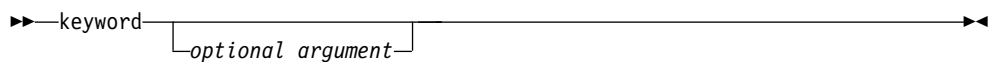
IBM XL Fortran extensions are marked by a number in the syntax diagram with an explanatory note immediately following the diagram.

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

- Required items are shown on the horizontal line (the main path):

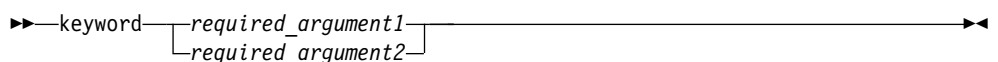


- Optional items are shown below the main path:



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:



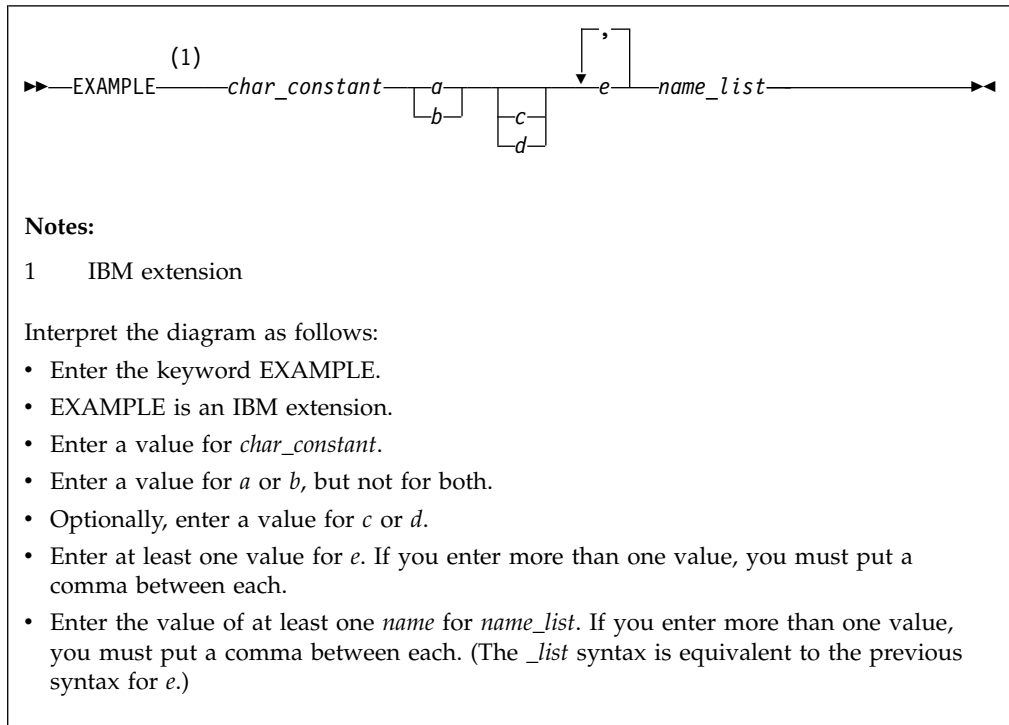
- The item that is the default is shown above the main path.



- 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:

- Quick Start Guide

The Quick Start Guide (`quickstart.pdf`) is intended to get you started with IBM XL Fortran for AIX, V15.1.3. It is located by default in the XL Fortran directory and in the `\quickstart` directory of the installation DVD.

- 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 DVD.

- 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, V15.1.3 Installation Guide*.

- Online product documentation

The fully searchable HTML-based documentation is viewable in IBM Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSGH4D_15.1.3/com.ibm.compilers.aix.doc/welcome.html.

- PDF documents
PDF documents are available on the web at <http://www.ibm.com/support/docview.wss?uid=swg27036673>.
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, V15.1.3 Installation Guide, SC27-4243-02</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, V15.1.3, SC27-4242-02</i>	getstart.pdf	Contains an introduction to the XL Fortran product, with information about setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
<i>IBM XL Fortran for AIX, V15.1.3 Compiler Reference, SC27-4244-02</i>	compiler.pdf	Contains information about the various compiler options and environment variables.
<i>IBM XL Fortran for AIX, V15.1.3 Language Reference, SC27-4245-02</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, V15.1.3 Optimization and Programming Guide, SC27-4246-02</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 Adobe Reader. If you do not have Adobe Reader, you can download it (subject to license terms) from the Adobe website at <http://www.adobe.com>.

More information related to XL Fortran, including IBM Redbooks® publications, white papers, and other articles, is available on the web at <http://www.ibm.com/support/docview.wss?uid=swg27036673>.

For more information about Fortran, see the Fortran café at <https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=b10932b4-0edd-4e61-89f2-6e478ccba9aa>.

Standards and specifications

XL Fortran is designed to support the following standards and specifications. You can refer to these standards and specifications 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.*
(This information uses its informal name, Fortran 90.)

- *Information technology - Programming languages - Fortran - Part 1: Base language, ISO/IEC 1539-1:1997.* (This information uses its informal name, Fortran 95.)
- *Information technology - Programming languages - Fortran - Part 1: Base language, ISO/IEC 1539-1:2004.* (This information uses its informal name, Fortran 2003.)
- *Information technology - Programming languages - Fortran - Part 1: Base language, ISO/IEC 1539-1:2010.* (This information uses its informal name, Fortran 2008. We currently provide partial support to this standard.)
- *Information technology - Further interoperability of Fortran with C, ISO/IEC TS 29113:2012.* (This information uses its informal name, Technical specification 29113, referred to as TS 29113. We currently provide partial support to this specification.)
- *Military Standard Fortran DOD Supplement to ANSI X3.9-1978, MIL-STD-1753* (United States of America, Department of Defense standard). Note that XL Fortran supports only those extensions documented in this standard that have also been subsequently incorporated into the Fortran 90 standard.
- *OpenMP Application Program Interface Version 3.1* (full support), and *OpenMP Application Program Interface Version 4.0* (partial support), 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*

Technical support

Additional technical support is available from the XL Fortran Support page at http://www.ibm.com/support/entry/portal/product/rational/xl_fortran_for_aix. 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 an email to compinfo@cn.ibm.com.

For the latest information about XL Fortran, visit the product information site at <http://www.ibm.com/software/products/en/xlfortran-aix>.

How to send your comments

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

Be sure to include the name of the manual, the part number of the manual, 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, V15.1.3 is an advanced, high-performance compiler that can be used for developing complex, computationally intensive programs, including interlanguage calls with C programs.

This section contains information about 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, V15.1.3 is part of a larger family of IBM C, C++, and Fortran compilers. XL Fortran, together with XL C and XL C/C++, comprises 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[®]/Q, and selected Linux distributions. The common code base, along with compliance with international programming language standards, helps support consistent compiler performance and ease of code portability across multiple operating systems and hardware platforms.

Operating system and hardware support

This section describes the operating systems and hardware that IBM XL Fortran for AIX, V15.1.3 supports.

IBM XL Fortran for AIX, V15.1.3 supports the following operating systems:

- AIX V6.1 TL 2 Service Pack 5 or later
- AIX V7.1
- AIX V7.2
- IBM i V7.1 PASE V7.1
- IBM i V7.2 PASE V7.2

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 run on POWER5, POWER5+, POWER6[®], POWER7[®], POWER7+[™], and POWER8[®] systems with the required software and disk space.

To exploit the various supported hardware configurations, the compiler provides options to tune the performance of applications according to the hardware type that runs 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 commands to invoke the compiler, for example, `xl`, `xl90`, `xl95`, `xl2003`, and `xl2008`. Compiler invocation commands are provided to support most standardized Fortran language levels and many popular language extensions.

The compiler also provides corresponding "`_r`" versions of most invocation commands, for example, `xl_r`. The "`_r`" invocations instruct the compiler to link and bind object files to threadsafe components and libraries, and produce threadsafe 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. You can benefit from using different options for the following tasks:

- Debugging your applications
- Optimizing and tuning application performance
- Selecting language levels and extensions for compatibility with nonstandard features and behaviors that are supported by other Fortran compilers
- Performing many other common tasks that would otherwise require changing the source code

You can 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.

If you frequently specify compiler option settings other than the default settings of XL Fortran, you can use makefiles to define your settings. Alternatively, you can create custom configuration files to define your own frequently used option settings.

For more information about using custom compiler configuration files, see "Using custom compiler configuration files" on page 43.

Utilization tracking configuration file

The utilization and reporting tool can be used to detect whether your organization's use of the compiler exceeds your license entitlements.

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 utilization tracking and reporting feature.

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

Language standard compliance

This topic describes the Fortran programming language specifications that IBM XL Fortran for AIX, V15.1.3 supports.

- Partial support for ISO/IEC TS 29113:2012 (referred to as the Technical Specification for further interoperability with C or TS 29113)
- Partial support for ISO/IEC 1539-1:2010 (referred to as Fortran 2008 or F2008)
- ISO/IEC 1539-1:2004 (referred to as Fortran 2003 or F2003)
- ISO/IEC 1539-1:1997 (referred to as Fortran 95 or F95)
- ISO/IEC 1539-1:1991(E) and ANSI X3.198-1992 (referred to as Fortran 90 or F90)
- ANSI X3.9-1978 (referred to as FORTRAN 77)

In addition to the standard language levels, XL Fortran supports the following language extensions:

- Partial support for OpenMP Application Program Interface V4.0
- OpenMP Application Program Interface V3.1
- Language extensions to support vector programming
- Common Fortran language extensions defined by other compiler vendors, in addition to those defined by IBM
- Industry extensions that are found in Fortran products from various compiler vendors
- Extensions specified in SAA Fortran

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

Source-code migration and conformance checking

XL Fortran provides 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 do not conform to the specified language level.

You can also use the `-qlanglvl` compiler option to specify a language level. If the language elements in your program source do not conform to the specified language level, the compiler issues diagnostic messages. Additionally, you can name your source files with common filename extensions such as `.f77`, `.f90`, `f95`, `.f03`, or `.f08`, and then use the generic compiler invocations such as `xlf` or `xlf_r` to automatically select the language level appropriate to the filename extension.

You can rebuild your FORTRAN 77, Fortran 90, Fortran 95, Fortran 2003, and Fortran 2008 source code with IBM XL Fortran for AIX, V15.1.3 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 "Compatibility with earlier versions" on page 17 for more information.

Related information

Tools, utilities, and commands

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

Tools

Utilization reporting tool

The utilization reporting tool generates a report describing your organization's utilization of the compiler. These reports help determine whether your organization's use of the compiler matches your compiler license entitlements. The **urt** command contains options that can be used to customize the report. For more information, see Tracking and reporting compiler usage in the *XL Fortran Compiler Reference*.

Utilities

CreateExportList utility

The **CreateExportList** utility creates a file that contains a list of all the global symbols found in a given set of object files. For more information, see Exporting symbols with the CreateExportList utility in the *XL Fortran Compiler Reference*.

Commands

genhtml command

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

Profile-directed feedback (PDF) related commands

cleanpdf command

The **cleanpdf** command removes all the PDF files or the specified PDF files from the directory to which profile-directed feedback data is written.

mergepdf command

The **mergepdf** command 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.

showpdf command

The **showpdf** command displays the following types of profiling information for all the procedures executed in a PDF run (compilation under the **-qpdf1** option):

- Block-counter profiling
- Call-counter profiling
- Value profiling
- Cache-miss profiling, if you specified the **-qpdf1=level=2** option during the **-qpdf1** phase.

You can view the first two types of profiling information in either text or XML format. However, you can view value profiling and cache-miss profiling information only in XML format.

For more information, see `-qpdf1`, `-qpdf2` in the *XL Fortran Compiler Reference*.

xlfndi The **xlfndi** script installs XL Fortran to a nondefault directory location. For more information, see Updating an advanced installation using **xlfndi** in the *XL Fortran Installation Guide*.

Program optimization

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

With these options, you can perform the following tasks:

- Select different levels of compiler optimizations.
- Control optimizations for loops, floating point, and other types of operations.
- Optimize a program for a particular class of machines or for a very specific machine configuration, depending on where the program will run.

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

- Reducing the number of instructions executed for critical operations
- Restructuring generated object code to make optimal use of the Power Architecture[®] processors
- Improving the usage of the memory subsystem
- Exploiting the ability of the architecture to handle large amounts of shared memory parallelization

Related information



Optimizing your applications



Optimization and tuning



Intrinsic procedures

64-bit object capability

The 64-bit object capability of the XL Fortran compiler 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 fit within a 64-bit address space, a separate 64-bit object format 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, 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 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 block of code. The existence of the directives in the source removes the need for the compiler to perform any dependence analysis on the parallel code. OpenMP directives require the presence of Pthread libraries to provide the necessary infrastructure for parallelization.

OpenMP directives address the following important issues of parallelizing an application:

1. Clauses and directives are available for scoping variables. Generally, variables should not be shared; that is, each thread 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 threads.
3. Directives are available to control synchronization between threads.

IBM XL Fortran for AIX, V15.1 supports OpenMP API Version 4.0 specification. For details, see “OpenMP 4.0” on page 28.

Related information



Optimizing your applications



The OpenMP API specification for parallel programming

Diagnostic reports

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

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

You can also obtain diagnostic information from the compiler in XML or HTML format. The XML and HTML reports provide information about optimizations that the compiler performed or could not perform. You can use this information to reduce programming effort when tuning applications, especially high-performance applications. The report is defined by an XML schema and is easily consumable by tools that you can create to read and analyze the results. For detailed information about this report and how to use it, see "Using reports to diagnose optimization opportunities" in the *XL Fortran Optimization and Programming Guide*.

Symbolic debugger support

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

The debugging information can be examined by **dbx** or any other symbolic debugger that supports the AIX XCOFF executable format to help you debug your programs.

Related information



-g



-qdbg

Chapter 2. What's new for IBM XL Fortran for AIX, V15.1.3

This section describes features and enhancements added to IBM XL Fortran for AIX, V15.1.3.

Language features

This topic lists new language features that are introduced in this release of XL Fortran.

► F2008

BIND attribute for an internal procedure

You can specify the **BIND** attribute without the **NAME=** specifier on an internal procedure. In previous releases, the **BIND** attribute could not be specified on an internal procedure.

DO CONCURRENT construct

With the **DO CONCURRENT** construct, you can specify that individual loop iterations have no interdependencies, in which case the execution order of the iterations can be indeterminate at the beginning of the execution of the **DO CONCURRENT** construct.

Polymorphic variable in an intrinsic assignment

In an intrinsic assignment *variable = expression*, *variable* now can be polymorphic. If *variable* is polymorphic, the following rules apply:

- *variable* must be allocatable.
- *variable* must be type-compatible with *expression*, or the declared types of *variable* and *expression* must conform.

Multiple allocate objects allowed on an ALLOCATE statement

You can allocate more than one *allocate_object* by using an **ALLOCATE** statement that contains the **SOURCE=** or **MOLD=** specifier. In previous releases, you could only allocate one *allocate_object* by using an **ALLOCATE** statement that contains the **SOURCE=** or **MOLD=** specifier.

VALUE attribute

You can specify the **VALUE** attribute on an array dummy argument that has either assumed shape or explicit shape. In the previous releases, you could not specify the **VALUE** attribute on array dummy arguments.

F2008 ◀




Related information in the XL Fortran Language Reference



Internal procedures



DO CONCURRENT construct (Fortran 2008)

-  Intrinsic assignment
-  ALLOCATE
-  VALUE (Fortran 2003)

Intrinsic procedures

This section describes intrinsic procedures that are new or changed for IBM XL Fortran for AIX, V15.1.3.

New intrinsic procedures

VEC_CIPHER_BE(ARG1,ARG2)

Performs one round of the AES cipher operation on an intermediate state ARG1 by using a given round key ARG2.

VEC_CIPHERLAST_BE(ARG1,ARG2)

Performs the final round of the AES cipher operation on an intermediate state ARG1 by using a given round key ARG2.

VEC_NCIPHER_BE(ARG1,ARG2)

Performs one round of the AES inverse cipher operation on an intermediate state ARG1 by using a given round key ARG2.

VEC_NCIPHERLAST_BE(ARG1,ARG2)

Performs the final round of the AES inverse cipher operation on an intermediate state ARG1 by using a given round key ARG2.

VEC_PMSUM_BE(ARG1,ARG2)

Performs an exclusive-OR operation by implementing a polynomial addition on each even-odd pair of the polynomial multiplication result of the corresponding elements.

VEC_SBOX_BE(ARG1)

Performs the SubBytes operation, as defined in *Federal Information Processing Standards FIPS-197*, on a given state ARG1.

VEC_SHASIGMA_BE(ARG1,ARG2,ARG3)

Performs a secure hash computation in accordance with *Federal Information Processing Standards FIPS-180-3*.

Changed intrinsic procedures

VEC_MULE(ARG1, ARG2)

VEC_MULE(ARG1, ARG2) now supports INTEGER(4) and UNSIGNED(4) vector types of parameters.

VEC_MULO(ARG1, ARG2)

VEC_MULO(ARG1, ARG2) now supports INTEGER(4) and UNSIGNED(4) vector types of parameters.

Other enhancements

This section describes other features and enhancements in IBM XL Fortran for AIX, V15.1.3.

Software License Metric (SLM) Tags logging support

IBM XL Fortran for AIX, V15.1.3 fully supports IBM Software License Metric (SLM) Tags logging so that you can use IBM License Metric Tool (ILMT) to track compiler license usage.

For more information, see "Tracking compiler usage with Software License Metric Tags logging" in the *XL Fortran Compiler Reference*.

Chapter 3. Migration of your applications

This section provides information about migrating your applications to IBM XL Fortran for AIX, V15.1.3.

The XL Fortran compiler helps you to port or to migrate source code among Fortran compilers by providing full Fortran 90, Fortran 95, Fortran 2003, and partial Fortran 2008 language support, and selected language extensions (including intrinsic functions and data types) from many different compiler vendors. Throughout this document, we refer to these extensions as “industry extensions”.

To protect your investment in FORTRAN 77 source code, you can easily invoke the compiler with a set of defaults that provide compatibility with earlier versions of XL Fortran. The `f77`, `fort77`, `xlf`, `xlf_r`, and `xlf_r7` commands provide maximum compatibility with existing FORTRAN 77 programs. The default options provided with the `f90`, `xlf90`, `xlf90_r`, and `xlf90_r7` commands give access to the full range of Fortran 90 language features. The default options provided with the `f95`, `xlf95`, `xlf95_r`, and `xlf95_r7` commands give access to the full range of Fortran 95 language features. The default options provided with the `f2003`, `xlf2003`, and `xlf2003_r` commands give access to the full range of Fortran 2003 language features. The default options provided with the `f2008`, `xlf2008`, and `xlf2008_r` commands give access to the Fortran 2008 language features supported in this release.

Additionally, you can name your source files with extensions such as `.f77`, `.f90`, `.f95`, `.f03`, or `.f08` and use the generic compiler invocations such as `xlf` or `xlf_r` to automatically select language-level appropriate defaults.

To protect your investments in FORTRAN 77 object code, you can link Fortran 90 and Fortran 95 programs with existing FORTRAN 77 object modules and libraries. See “Linking new objects with existing ones” on page 51 for details.

More advice is provided in the following sections to help make the transition from an earlier version of the XL Fortran compiler as fast and simple as possible.

Related information

- Chapter 2, “What’s new for IBM XL Fortran for AIX, V15.1.3,” on page 9

Things to note in IBM XL Fortran for AIX, V15.1.3

Because IBM XL Fortran for AIX, V15.1.3 is highly compatible with XL Fortran Versions 14 through 3 inclusive, most of the advice in this section applies to upgrades from Version 2, or earlier levels of XL Fortran.

- The `xlf90`, `xlf90_r`, `xlf90_r7`, and `f90` commands provide Fortran 90 conformance. The `xlf95`, `xlf95_r`, `xlf95_r7`, and `f95` commands provide Fortran 95 conformance. The `xlf2003`, `xlf2003_r`, and `f2003` commands provide Fortran 2003 conformance. The `xlf2008`, `xlf2008_r`, and `f2008` commands provide partial Fortran 2008 conformance. However, these commands may cause some problems with existing FORTRAN 77 programs. The `xlf`, `xlf_r`, `xlf_r7`, `f77`, and `fort77` commands avoid some of these problems by keeping the old behavior wherever possible.

- Fortran 90 introduced the idea of kind parameters for types. Except for the types complex and character, XL Fortran uses numeric kind parameters that correspond to the lengths of the types. For the type complex, the kind parameter is equal to the length of the real portion, which is half of the overall length. For the type character, the kind parameter is equal to the number of bytes that are required to represent each character, and this value is 1. A FORTRAN 77 declaration that is written using the * extension for length specifiers can now be rewritten with a kind parameter:

```

INTEGER*4 X ! F77 notation with extension.
INTEGER(4) X ! F90 standard notation.
COMPLEX*8 Y ! *n becomes (n) for all types except
COMPLEX(4) Y ! COMPLEX, where the value is halved.

```

This new form is the one that is used consistently throughout the XL Fortran manuals.

Because the values of kind parameters may be different for different compilers, you may want to use named constants, placed in an include file or a module, to represent the kind parameters used in your programs. The **SELECTED_CHAR_KIND**, **SELECTED_INT_KIND** and **SELECTED_REAL_KIND** intrinsic functions also let you determine kind values in a portable way.

- Fortran 90 introduced a standardized free source form for source code, which is different from the XL Fortran Version 2 free source form. The **-qfree** and **-k** options now use the Fortran 90 free source form; the Version 2 free source form is available through the option **-qfree=ibm**.
- The library that provides Fortran 90, Fortran 95, Fortran 2003, and partial Fortran 2008 support is **libxlf90.a**, located in `/usr/lib`. A **libxlf.a** library of stub routines is provided in `/usr/lib`, but it is only used for linking existing Version 1 or 2 object files or running existing executables. When a Version 1 or Version 2 object file calls entry points in **libxlf.a**, those entry points then call equivalent entry points in **libxlf90.a**. If you recompile such object files, the result could be improved I/O performance, because the entry points in **libxlf90.a** are called directly.

Avoiding or fixing upgrade problems

Although XL Fortran is generally compatible with FORTRAN 77 programs, there are some changes in XL Fortran and the Fortran 90, Fortran 95, Fortran 2003, and Fortran 2008 languages that you should be aware of.

To preserve the behavior of existing compilation environments, the **xlf** and **f77** commands both work as they did in earlier XL Fortran versions wherever possible. As you write entirely new Fortran 90, Fortran 95, Fortran 2003, or Fortran 2008 programs or adapt old programs to avoid potential problems, you can begin using the **xlf90**, **xlf95**, **xlf2003**, and **xlf2008** commands, which use Fortran 90, Fortran 95, Fortran 2003, and Fortran 2008 conventions for source-code format.

Note that in the following table, you can substitute **xlf_r** or **xlf_r7** for **xlf**, **xlf90_r** or **xlf90_r7** for **xlf90**, **xlf95_r** or **xlf95_r7** for **xlf95**, **xlf2003_r** for **xlf2003**, and **xlf2008_r** for **xlf2008**.

*Table 4. Potential problems migrating programs to XL Fortran V15.1.3. The column on the right shows which problems you can avoid by using the **xlf** or **f77** command.*

Potential Problem	Solution or Workaround	xlf Avoids?
Compilation Problems		

Table 4. Potential problems migrating programs to XL Fortran V15.1.3 (continued). The column on the right shows which problems you can avoid by using the **xlf** or **f77** command.

Potential Problem	Solution or Workaround	xlf Avoids?
New intrinsic procedure names may conflict with external procedure names. The intrinsic procedure is called instead of the external procedure.	Use the -qextern option, or insert EXTERNAL statements to avoid the ambiguity. Consider switching to the Fortran 90 or Fortran 95 procedure if it does what you want.	
The .XOR. intrinsic operator is not recognized.	Use the option -qxlf77=intxor.	✓
Zero-sized objects are not allowed by the compiler.	Use the xlf90 or xlf95 command, or use the -qzerosize option with the xlf or f77 command.	
Performance / Optimization Problems		
Existing programs or programs linked with older XL Fortran object files run more slowly or do not show expected performance improvements on new hardware.	Recompile everything.	
Programs compiled with -03 or -qhot optimization behave differently from those unoptimized (different results, exceptions, or compilation messages).	Try adding the -qstrict option.	
The option combination -0 and -1 cannot be abbreviated to -01 , to avoid misunderstandings. (There are -02 , -03 , -04 , and -05 optimization levels, but there is no -01 .)	Specify -0 and -1 as separate options.	
Programs that use integer POINTERS produce incorrect results when optimized.	Specify the option -qalias=intptr with the xlf90 or xlf95 command, or use the xlf command.	✓
Runtime problems		
Programs that read to the end of the file and then try to append records without first executing a BACKSPACE statement do not work correctly. The write requests generate error messages.	To compile existing programs, specify the option -qxlf77=softeof with the xlf90 or xlf95 command, or use the xlf command. For new programs, add the BACKSPACE statement before writing past the endfile record.	✓
Uninitialized variables are not necessarily set to zero, and programs that ran before may exceed the user stack limit. The reason is that the default storage class is now AUTOMATIC , rather than STATIC (an implementation choice allowed by the language).	Ensure that you explicitly initialize your variables, use the -qsave option with the xlf90 or xlf95 command, or add SAVE statements where needed in the source.	✓

Table 4. Potential problems migrating programs to XL Fortran V15.1.3 (continued). The column on the right shows which problems you can avoid by using the `xl` or `f77` command.

Potential Problem	Solution or Workaround	xl Avoids?
Writing data to some files opened without a POSITION= specifier overwrites the files, instead of appending the data.	Use the option -qposition=appendold , or add POSITION= specifiers where needed.	✓
Newly compiled programs are unable to read existing data files containing NAMELIST data. The reason is that the Fortran 90 and Fortran 95 standards define a namelist format that is different from that used on AIX in the past.	Set the environment variable XLFRTEOPTS to the string namelist=old . The programs that produced the old NAMELIST data must be recompiled.	
Some I/O statements and edit descriptors accept or produce slightly different input and output. For example, real output now has a leading zero when appropriate. The changes to I/O formats are intended to be more usable and typical of industry practice, so you should try to use the defaults for any new data you produce.	When you need to maintain compatibility with existing data files, compile with the xl command. If the incompatibility is due to a single specific I/O change, see if the -qx1f77 option has a suboption for compatibility with earlier versions. If so, you can switch to the xl90 or xl95 command and use the -qx1f77 option on programs that use the old data files.	✓
Numeric results and I/O output are not always exactly identical with XL Fortran Version 2. Certain implementation details of I/O, such as spacing in list-directed output and the meanings of some IOSTAT values, have changed since XL Fortran Version 2. (This entry is similar to the previous one except that these differences are not compatible with earlier versions.)	You may need to generate existing data files again or to change any programs that depend on these details. When compatibility with earlier versions is not provided by the -qx1f77 compiler option or XLFRTEOPTS runtime options, there is no way to get the old behavior back.	
SIGN(A,B) now returns $- A $ when B = -0.0 . Prior to XL Fortran Version 7.1, it returned $ A $.	This behavior conforms with the Fortran 95 standard and is consistent with the IEEE standard for binary floating-point arithmetic. It occurs because the -qx1f90=signedzero option is turned on. Turn it off, or specify a command that does not use this option by default.	✓

Table 4. Potential problems migrating programs to XL Fortran V15.1.3 (continued). The column on the right shows which problems you can avoid by using the `xlf` or `f77` command.

Potential Problem	Solution or Workaround	xlf Avoids?
A minus sign is printed for a negative zero in formatted output. A minus sign is printed for negative values that have an outputted form of zero (that is, in the case where trailing non-zero digits are truncated from the output so that the resulting output looks like zero). Prior to XL Fortran Version 7.1, minus signs were not printed in these situations.	This behavior conforms with the Fortran 95 standard and occurs because the <code>-qxlf90=signedzero</code> option is turned on. Turn it off, or specify a command that does not use this option by default.	✓
The handling of IEEE infinity and not-a-number (NaN) exceptional values has changed. By default, XL Fortran displays IEEE infinity and NaN exceptional values based on the command used to compile the source code. This can result in a mixture of IEEE exceptional value outputs if there are multiple object files that were compiled with different options.	XL Fortran allows control over the display of IEEE infinity and NaN exceptional values. To get the previous behavior, set the <code>XLFRTEOPTS</code> environment variable to the <code>naninfoutput=old</code> string.	✓

Compatibility with earlier versions

This section describes issues about compatibility with earlier versions and their workarounds.

-qsave and -qxlf77=persistent no longer enabled by threadsafe invocation commands

To reduce the risk of thread safety coding issues, the threadsafe invocation commands `xlf_r` and `xlf_r7` no longer list `-qsave` and `-qxlf77=persistent` as default options in the compiler configuration file. The `-qsave` option specifies that the default storage class for local variables is static. The `-qxlf77=persistent` option saves the address of arguments to subprograms with `ENTRY` statements in static storage.

If you compile a multithreaded program that has uninitialized local variables with `xlf_r` or `xlf_r7`, the program might fail execution. When `-qsave` is in effect, local variables are stored in static storage and have an initial value of 0. Because `xlf_r` and `xlf_r7` no longer imply `-qsave`, local variables are stored in automatic storage and do not have a particular initial value.

To ensure proper program behavior, always explicitly initialize the local variables in your program or specify `-qinitauto` so that the local variables are initialized by the compiler. Alternatively, specify `-qsave` or `-qxlf77=persistent` if you are sure that they are safe for your program. If you specify `-qsave` or `-qxlf77=persistent` with `xlf_r` or `xlf_r7`, a compiler warning message about thread safety is issued.

OpenMP threadprivate data compatibility issue between V13.1 and its earlier versions

Starting from XL Fortran V13.1, the implementation of the threadprivate data, that is, OpenMP threadprivate variable, has been improved. The operating system

thread local storage is used instead of the runtime implementation. The new implementation might improve performance on some applications.

If you plan to mix the object (.o) files that you have compiled with levels prior to 13.1 with the object files that you compiled with XL Fortran, and the same OpenMP threadprivate variables are referenced in both old and new object files, different implementations might cause incompatibility issues. A link error, a compile time error or other undefined behaviors might occur. To support compatibility with earlier versions, you can use the `-qsmp=noostls` suboption to switch back to the old implementation. You can recompile the entire program with the default suboption `-qsmp=ostls` to get the benefit of the new implementation.

If you are not sure whether the object files you have compiled with levels prior to XL Fortran contain any old implementation, you can use the `nm` command to determine whether you need to use the `-qsmp=noostls` suboption. The following code is an example that shows how to use the `nm` command:

```
> nm oldfiles.o
...
._xlGetThStorageBlock U          -
._xlGetThValue         U          -
...
```

In the preceding example, if `_xlGetThStorageBlock` or `_xlGetThValue` is found, this means the object files contain old implementation. In this case, you must use `-qsmp=noostls`; otherwise, use the default suboption `-qsmp=ostls`.

Binary compatibility issue between V11.1 or V12.1 and its later releases

Because of a change to the signature of a compiler-generated routine, mixing code compiled with XL Fortran V11.1 or V12.1 with code containing parameterized derived types compiled with the latest compiler may require recompiling all the source files. Otherwise, a runtime error will occur if the older code uses certain polymorphic references that can resolve to parameterized derived type objects where a length parameter is involved.

For example, the new application extends a derived type whose declaration was compiled using either XL Fortran V11.1 or XL Fortran V12.1, and the extending type has length derived type parameters, and the new application passes an object that has a dynamic type of the extending type through polymorphism via argument association or function reference to a procedure compiled using XL Fortran V11.1 or XL Fortran V12.1 compiler version.

XL Fortran runtime detects the above issue and halts the execution with the following error message:

```
XL Fortran detected a mismatch in a compiler-generated routine. If your
code contains compilation units compiled with XL Fortran V11.1 or V12.1,
recompile them with the latest XL Fortran compiler.
```

Example of argument association:

V11.1 or V12.1 release	V13.1 or later releases
<pre> module m type base integer i end type contains subroutine sub(arg) class(base) :: arg class(base), allocatable :: local allocate(local, source=arg) end subroutine end module </pre>	<pre> use m type, extends(base) :: child(1) integer, len :: l integer :: m(1) integer :: n(1) end type class(base), allocatable :: b1 allocate(child(5) :: b1) call sub(b1) end </pre>

The problem is the ALLOCATE statement in 11.1/12.1 code is calling a compiler-generated routine that is compiled using newer releases of XL Fortran compiler other than XL Fortran V11.1 or XL Fortran V12.1 through type bound procedure call. Since the signature of the compiler-generated routine is different between 11.1/12.1 releases and newer releases, module m needs to be recompiled using newer releases of XL Fortran compiler.

Example of function call:

V11.1 or V12.1 release	V13.1 or later releases
<pre> module m type base integer i contains procedure :: bar => bar_base end type type container class(base), allocatable :: b1 end type contains subroutine bar_base(a) class(base) a print *, "base" end subroutine end module program main use m interface function foo() import type(container) :: foo end function end interface type(container) :: c1 c1 = foo() call c1%b1%bar end program </pre>	<pre> module n use m type, extends(base) :: child(1) integer, len :: l integer a(1) contains procedure :: bar => bar_child end type contains subroutine bar_child(a) class(child(*)) a print *, "child" print *, a%a end subroutine end module function foo() use n type(container) :: foo allocate(child(6) :: foo%b1) end function </pre>

The problem is `c1 = foo()` in program main is calling a compiler-generated routine that is compiled using newer releases of XL Fortran compiler other than V11.1 or V12.1 through type bound procedure call via function foo. Since the signature of the compiler-generated routine is different between 11.1/12.1 releases and newer

releases, program main needs to be recompiled using newer releases of XL Fortran compiler.

Chapter 4. Enhancements added in earlier releases

This section describes enhancements added in earlier releases. These enhancements also apply to the current release.

Enhancements added in Version 15.1.2

This section describes features and enhancements added in IBM XL Fortran for AIX, V15.1.2. These features and enhancements apply to later releases as well.

Fortran 2008 features

This topic lists the Fortran 2008 features that are introduced in this release of XL Fortran.

New restriction on elemental procedures

If a dummy argument of an elemental procedure does not have the **VALUE** attribute, the dummy argument must have the **INTENT** attribute specified.

New restriction on the reference to an elemental function

In a reference to an elemental procedure, if any actual argument is an array, every actual argument that corresponds to an **INTENT(OUT)** or **INTENT(INOUT)** dummy argument must be an array.

Language interoperability features

XL Fortran implements selected language interoperability features, which accept programs that contain parts written in Fortran and parts written in the C language.

This version of XL Fortran provides support for the following language interoperability features as specified in TS 29113:

Assumed-length arguments of type character

BIND(C) procedures that have nonallocatable and nonpointer dummy arguments of type character with assumed length can interoperate with C functions having formal parameters that are pointers to type `CFI_cdesc_t`.

The **C_PTRDIFF_T** named constant in the **ISO_C_BINDING** module

The **C_PTRDIFF_T** named constant is added to the **ISO_C_BINDING** module. Fortran entities of type **INTEGER(C_PTRDIFF_T)** are interoperable with C entities of type `ptrdiff_t`.

Relaxed restrictions on module procedure **C_FUNLOC(X)** and **C_LOC(X)** of intrinsic module **ISO_C_BINDING**

Some restrictions on module procedure **C_FUNLOC(X)** and **C_LOC(X)** of intrinsic module **ISO_C_BINDING** that are put forward by earlier Fortran language standards are removed according to TS29113. For details, see **C_FUNLOC(X)** and **C_LOC(X)** in the *XL Fortran Language Reference* and .

Intrinsic procedures

This section describes intrinsic procedures that are new for IBM XL Fortran for AIX, V15.1.2.

STORAGE_SIZE (A, KIND)

Returns the storage size in bits for an element of an array that has the dynamic type and type parameters of A.

VEC_MERGEE(ARG1,ARG2)

Merges the values of even-numbered elements of two vectors.

VEC_MERGEO(ARG1,ARG2)

Merges the values of odd-numbered elements of two vectors.

VEC_REVB(ARG1)

Returns a vector that contains the bytes of the corresponding element of the argument in the reverse byte order.

VEC_REVE(ARG1)

Returns a vector that contains the elements of the argument in the reverse element order.

VEC_XL(ARG1, ARG2)

Loads a 16-byte vector from the memory address specified by the displacement ARG1 and the pointer ARG2.

VEC_XL_BE(ARG1, ARG2)

Loads a 16-byte vector from the memory address specified by the displacement ARG1 and the pointer ARG2.

VEC_XST(ARG1, ARG2, ARG3)

Stores the elements of the 16-byte vector ARG1 to the effective address obtained by adding the displacement provided in ARG2 with the address provided by ARG3. The effective address is not truncated to a multiple of 16 bytes.

VEC_XST_BE(ARG1, ARG2, ARG3)

Stores the elements of the 16-byte vector ARG1 in big endian element order to the effective address obtained by adding the displacement provided in ARG2 with the address provided by ARG3. The effective address is not truncated to a multiple of 16 bytes.

Related information



Intrinsic procedures

Commands

This section describes new, changed, or removed compiler commands.

resetpdf

This command has been removed. It is recommended that you use the **cleanpdf** command instead. The behavior of the **resetpdf** command is the same as that of the **cleanpdf** command. For more information, see `-qpdf1`, `-qpdf2` in the *XL Fortran Compiler Reference*.

Compiler options

This topic describes new or changed compiler options.

-qfloat

The following suboptions are added:

subnormals

This suboption asserts to the compiler that the code uses subnormal floating point values, also known as denormalized floating point values.

nosubnormals

This suboption asserts to the compiler that the code does not use subnormal floating point values, also known as denormalized floating point values.

Whether or not you specify this suboption, the behavior of your program will not change, but the compiler uses this information to gain possible performance improvements. The suboptions take effect only on POWER8 processors. To use **-qfloat=subnormals** or **-qfloat=nosubnormals**, you must also specify the **-qarch=pwr8** and **-qtune=pwr8** options.

-MMD

Produces a dependency output file containing targets suitable for inclusion in a description file for the **make** command. This option was named as **-M** in XL Fortran for AIX, V15.1.

-qfmt

Sets the byte order for I/O operations on unformatted data files.

-qvisibility

Specifies the visibility attribute for external linkage symbols in object files.

Other XL Fortran updates

This section describes other updates added in IBM XL Fortran for AIX 15.1.2.

►TS 29113

Passing scalar actual arguments that correspond to assumed-size dummy arguments of assumed-type

You can pass a scalar as the actual argument that corresponds to an assumed-size dummy argument of assumed-type. An example is provided in Assumed-size arrays in the *XL Fortran Language Reference*.

TS 29113◄

The CONVERT specifier in the OPEN and INQUIRE statements

You can use **CONVERT= *char_expr*** in the **OPEN** statement to set the byte order for I/O operations on unformatted data files. *char_expr* is a scalar character expression whose value must evaluate to **NATIVE**, **BIG_ENDIAN**, or **LITTLE_ENDIAN**. You can also use **CONVERT= *char_var*** in the **INQUIRE** statement to find out the byte order for I/O operations on unformatted data files. For details, see **OPEN** and **INQUIRE** in the *XL Fortran Language Reference* and .

Enhancements added in Version 15.1

This section describes features and enhancements added to the compiler in Version 15.1. These features and enhancements apply to later versions as well.

Support for POWER8 processors

XL Fortran for AIX, V15.1 supports POWER8 processors.

The new features and enhancements introduced in support of the POWER8 processors, fall under the following categories:

- MASS libraries for POWER8 processors
- Compiler options for POWER8 processors
- Hardware directives and intrinsics for POWER8 processors

Mathematical Acceleration Subsystem (MASS) libraries for POWER8 processors

Scalar libraries

The MASS library interfaces include the following features:

- The scalar functions have generic interfaces that can be called with **REAL(4)** or **REAL(8)** arguments.
- The scalar functions are marked pure. You can call them from pure procedures.
- The scalar functions are marked elemental. You can call them with an array argument and apply them to all the array elements.
- The intent of the argument is specified to assist in compiler error checking.

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

Vector libraries

The vector MASS library **libmassvp8.a** contains vector procedures that have been tuned for the POWER8 architecture. The procedures can be used in either 32-bit mode or 64-bit mode.

The MASS vector library interfaces include the following features:

- The vector functions have generic interfaces that can be called with **REAL(4)** or **REAL(8)** arguments.
- The vector functions are marked pure. You can call them from pure procedures.
- The intent of the argument is specified to assist in compiler error checking.

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_simdp8.a** contains an accelerated set of frequently used math intrinsic procedures that provide improved performance over the corresponding standard system library procedures.

The MASS SIMD library interfaces include the following features:

- The SIMD functions are marked pure. You can call them from pure procedures.
- The intent of the argument is specified to assist in compiler error checking.

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

Compiler options for POWER8 processors

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.

The new **-qarch=pwr8** suboption produces object code containing instructions that will run on the POWER8 hardware platforms. With the new **-qtune=pwr8** suboption, optimizations are tuned for the POWER8 hardware platforms.

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

Hardware directives and intrinsics for POWER8 processors

New hardware directives and intrinsics are added to support the following POWER8 processor features:

- POWER8 intrinsics for vector processing
- POWER8 cryptography intrinsics
- POWER8 transactional memory intrinsics
- POWER8 prefetch directives
- POWER8 prefetch intrinsic procedures

For more information about the directives and intrinsic procedures, see Hardware-specific directives, Hardware-specific intrinsic procedures (IBM extension), Vector intrinsic procedures (IBM extension) , or The TRANSACTIONAL_MEMORY intrinsic module (IBM extension) in the *XL Fortran Language Reference*.

Fortran 2008 features

XL Fortran implements selected features of the Fortran 2008 standard.

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

- **BACK=** arguments in **MAXLOC** and **MINLOC**
- Double colon separators (::) in **PROCEDURE** statements
- Intrinsic procedures for manipulating bits through combined shifting, merging, masking, or shifting
- Extensions to the generic resolution rules
- **FINDLOC** intrinsic procedure
- Impure elemental procedures
- Separate module subprograms
- Submodules
- The **MODULE** prefix specifier
- Type specification in the **FORALL** statement and construct

BACK= arguments in the MAXLOC and MINLOC intrinsic procedures

You can specify the search direction in the **MAXLOC** and **MINLOC** intrinsic procedures with the **BACK=** argument keyword.

Double colon separators in PROCEDURE statements

You can optionally use a double colon separator (::) in **PROCEDURE** and **MODULE PROCEDURE** statements inside interface blocks.

Intrinsic procedures for bit manipulations

You can use the following intrinsic procedures for manipulating bits through combined shifting, merging, masking, or shifting:

- DSHIFTL
- DSHIFTR
- MASKL
- MASKR
- MERGE_BITS
- SHIFTA
- SHIFTL
- SHIFTR

Extensions to the generic resolution rules

The Fortran 2008 standard extends the generic resolution rules to distinguish between allocatable and pointer dummy arguments and between procedure and data dummy arguments. For details, see Unambiguous generic procedure references in the *XL Fortran Language Reference*.

FINDLOC intrinsic procedure

The **FINDLOC** intrinsic procedure locates the element of an array whose value equals the target value under the condition that is specified by parameters. It returns the subscript of the element using positive integers. For details, see **FINDLOC**(ARRAY, VALUE, DIM, MASK, KIND, BACK) in the *XL Fortran Compiler Reference*.

Impure elemental procedures

Elemental procedures are no longer required to be pure in Fortran 2008. You can explicitly declare procedures with the **IMPURE** prefix specifier.

Separate module subprograms

A separate module subprogram defines a separate module procedure that is declared by a corresponding module procedure interface body. For details, see Separate module subprograms and Separate module procedures.

Submodules

A submodule extends a module or another submodule. You can declare a module procedure interface body in a module and implement it as a separate module procedure in one of the descendant submodules. The submodule feature provides the following benefits:

- If only the implementation of a separate module procedure is changed, but the interface remains the same, you do not need to recompile the file that contains the module in which the corresponding module procedure interface body is declared.

- Two submodules of different modules can access the ancestor module of each other through use association without causing circular dependency.
- You can put entities in the intermediate submodule level so that the entities are shared by the descendant submodules. If some of the entities are changed, the interpretation of anything that is accessible from the ancestor module by use association is not affected. This also prevents cascades of reprocessing and testing.

For details about the syntax and rules, see Submodules in the *XL Fortran Language Reference*. For details about the benefits of submodules and an example that demonstrates how to use submodules, see Submodules in the *XL Fortran Optimization and Programming Guide*.

The **MODULE** prefix specifier

To declare a module procedure interface body or define a separate module procedure, specify the **MODULE** prefix specifier for the **FUNCTION** or **SUBROUTINE** statement. For details, see **FUNCTION** and **SUBROUTINE**.

Type specification in the **FORALL** statement and construct

You can optionally include type specifications for index variables in the **FORALL** statement and construct.

Language interoperability features

XL Fortran implements selected language interoperability features, which accept programs that contain parts written in Fortran and parts written in the C language.

This version of XL Fortran provides support for the following language interoperability features as specified in TS 29113:

- Interoperability of assumed-rank arguments
- Interoperability of assumed-type arguments
- Interoperable procedures with dummy arguments that have **ALLOCATABLE**, **OPTIONAL**, or **POINTER** attributes
- Interoperable variables in asynchronous communication
- The `ISO_Fortran_binding.h` header file
- The **RANK** intrinsic procedure

Interoperability of assumed-rank arguments

Assumed-rank arguments are introduced to facilitate the interoperability with C functions that accept arguments of arbitrary rank. For details, see Interoperability of assumed-rank arguments in the *XL Fortran Language Reference*.

Interoperability of assumed-type arguments

Assumed-type arguments are introduced to facilitate the interoperability with formal parameters of type `void*` in C functions. For details, see Assumed-type objects in the *XL Fortran Language Reference*.

Interoperable procedures with allocatable, optional, and pointer dummy arguments

You can specify **ALLOCATABLE**, **OPTIONAL**, and **POINTER** attributes for a dummy argument in a procedure interface that has the **BIND(C)** attribute. For details, see *Optional arguments and Allocatable and pointer arguments in the XL Fortran Language Reference*.

Interoperable variables in asynchronous communication

Asynchronous communication for a Fortran variable can occur when procedures that are defined by means other than Fortran are called. You must specify the **ASYNCHRONOUS** attribute for the variables that are used for the asynchronous communication. For details, see *Interoperable variables in asynchronous communication in the XL Fortran Language Reference*.

The ISO_Fortran_binding.h header file

By using a C descriptor whose type is defined in the `ISO_Fortran_binding.h` header file, you can pass a Fortran data object to C. By using functions that are defined in the `ISO_Fortran_binding.h` header file, you can also manipulate a Fortran data object in C. For details, see *The ISO_Fortran_binding.h header file in the XL Fortran Language Reference*.

The RANK intrinsic procedure

You can use the **RANK** intrinsic procedure to get the rank of a data object, such as an assumed-rank object. For details, see `RANK(A)` in the *XL Fortran Language Reference*.

OpenMP 4.0

XL Fortran for AIX, V15.1 partially supports the OpenMP Application Program Interface Version 4.0 specification. The XL Fortran implementation is based on IBM's interpretation of the OpenMP Application Program Interface Version 4.0.

This version of XL Fortran supports the following OpenMP 4.0 features:

- capture clause enhancements
- `OMP_DISPLAY_ENV` environment variable

update and capture clauses enhancements

The update and capture clauses of the `atomic` construct are extended to support more expression forms.

capture clause enhancements

The capture clause of the `atomic` construct is extended to support more syntax forms.

OMP_DISPLAY_ENV environment variable

You can use the `OMP_DISPLAY_ENV` environment variable to display the values of the internal control variables (ICVs) associated with the environment variables and the build-specific information about the runtime library.

Related information

- "Parallel programming with XL Fortran" in the *XL Fortran Optimization and Programming Guide*
- The OpenMP API specification for parallel programming

Directives and intrinsic procedures

The following major categories of directives and intrinsic procedures are new to this release.

Note: POWER8 directives and intrinsic procedures are valid only when `-qarch=pwr8` is set or implied.

POWER8 intrinsic procedures for vector processing

The following vector intrinsic procedures are added:

- The vector gather bits by bytes doubleword procedure
 - VEC_GBB
- The vector count leading zeros procedure
 - VEC_CNTLZ
- The vector population count procedure
 - VEC_POPCNT
- Extended vector logical operations procedures
 - VEC_EQV
 - VEC_NAND
 - VEC_ORC
- 128-bit integer add and subtract procedures
 - VEC_ADD_U128
 - VEC_SUB_U128
 - VEC_ADDE_U128
 - VEC_SUBE_U128
 - VEC_ADDC_U128
 - VEC_SUBC_U128
 - VEC_ADDEC_U128
 - VEC_SUBEC_U128
 - VEC_BPERM

The following intrinsic procedures are extended to support doubleword types:

- Vector pack procedures
 - VEC_PACK
 - VEC_PACKS
 - VEC_PACKSU
- Vector unpack procedures
 - VEC_UNPACKL
 - VEC_UNPACKH
- Vector add and subtract procedures
 - VEC_ADD
 - VEC_SUB

- Vector max and min procedures
 - VEC_MAX
 - VEC_MIN
- Vector shift and rotate procedures
 - VEC_RL
 - VEC_SL
 - VEC_SR
 - VEC_SRA
- Vector compare procedures
 - VEC_CMPGE
 - VEC_CMPLE

POWER8 cryptography intrinsic procedures

The following intrinsic procedures are provided to perform cryptographic operations:

- Advanced Encryption Standard (AES) procedures
 - VCIPHER
 - VCIPHERLAST
 - VNCIPHER
 - VNCIPHERLAST
 - VSBOX
- Secure Hash Algorithm (SHA) procedures
 - VSHASIGMAD
 - VSHASIGMAW
- Miscellaneous procedures
 - VPMSUMB
 - VPMSUMH
 - VPMSUMW
 - VPMSUMD
 - VPERMXOR

POWER8 transactional memory intrinsic procedures

Transactional memory is a model for parallel programming. In this model, you can designate a block of instructions or statements to be treated atomically.

The `transactional_memory` module provides the following intrinsic procedures to work with transactions:

- Transaction begin and end procedures
 - TM_BEGIN
 - TM_END
 - TM_SIMPLE_BEGIN
- Transaction abort procedures
 - TM_ABORT
 - TM_NAMED_ABORT
- Transaction inquiry procedures

- TM_FAILURE_ADDRESS
- TM_FAILURE_CODE
- TM_IS_CONFLICT
- TM_IS_FAILURE_PERSISTENT
- TM_IS_FOOTPRINT_EXCEEDED
- TM_IS_ILLEGAL
- TM_IS_NAMED_USER_ABORT
- TM_IS_NESTED_TOO_DEEP
- TM_IS_USER_ABORT
- TM_NESTING_DEPTH

POWER8 prefetch directives

The following directives display the problem state control of the Data Stream Control Register (DSCR) in an intuitive, portable, and optimization-friendly way:

- Transient attribute enable directives
 - HARDWARE_TRANSIENT_ENABLE
 - LOAD_TRANSIENT_ENABLE
 - SOFTWARE_TRANSIENT_ENABLE
 - STORE_TRANSIENT_ENABLE
- Unit count enable and set directives
 - HARDWARE_UNIT_COUNT_ENABLE
 - SET_PREFETCH_UNIT_COUNT
 - SOFTWARE_UNIT_COUNT_ENABLE
- Prefetch depth directives
 - DEFAULT_PREFETCH_DEPTH
 - DEPTH_ATTAINMENT_URGENCY
- Load stream enable and disable directives
 - LOAD_STREAM_DISABLE
 - STRIDE_N_STREAM_ENABLE

POWER8 prefetch intrinsic procedures

You can use the following intrinsic procedures to get or set the value of the DSCR:

- PREFETCH_GET_DSCR_REGISTER
- PREFETCH_SET_DSCR_REGISTER

Related information:

- Hardware-specific directives,
- Hardware-specific intrinsic procedures (IBM extension)
- Vector intrinsic procedures (IBM extension)
- The TRANSACTIONAL_MEMORY intrinsic module (IBM extension)
- *XL Fortran Language Reference*
- -qarch

Compiler options

This topic describes new or changed compiler options.

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

- I** This option is extended to support submodules. You can use it to add a directory to the search path for submodule symbol (.smod) files.
- M, -qmakedep** This option produces a dependency output file containing targets suitable for inclusion in a description file for the **make** command.
- MF** This option is used to specify the name or location for the dependency output files that are generated by the **-qmakedep** or **-M** option.
- MT** This option is used to specify the target name of the object file in the **make** rule in the dependency output file that is generated by the **-qmakedep** or **-M** option.
- qarch** The option default is updated to **pwr4**. Suboptions denoting old hardware families are silently upgraded to newer architectures.

The following suboptions are added or updated:

-qarch=pwr7

This suboption produces object code containing instructions that run on the POWER7, POWER7+, or POWER8 hardware platforms.

-qarch=pwr8

This suboption produces object code containing instructions that run on the POWER8 hardware platforms.

-qcheck

The following suboptions are added:

-qcheck=all

This suboption enables all suboptions.

-qcheck=bounds

This suboption checks each reference to an array, array section, or character substring to ensure that the reference stays within the defined bounds of the entity.

-qcheck=stacklobber

This suboption detects a certain type of stack corruption in your programs.

-qcheck=unset

This suboption checks for automatic variables that are used before they are set at run time.

-qdbfmt=dwarf4

This suboption generates debugging information in DWARF 4 format.

-qfunctrace

This option is extended to support submodules.

-qhelp This option displays the man page of the compiler.

-qinfo

The following suboptions are added :

-qinfo=HOSTASSOCIATION

This suboption notifies you about entities that are accessed by host association.

-qinfo=mt

This suboption notifies you about potential places where synchronization is needed.

-qinfo=unset

This suboption detects automatic variables that are used before they are set, and flags them with informational messages at compile time.

-qmoddir

This option is extended to support submodules. You can use it to specify the location for any submodule (.smo) symbol files.

-qpath This option specifies locations for compiler components. You can specify a different path for each component.

-qpdf1=unique

This suboption creates a unique PDF file for each process during run time.

-qprefetch=dscr

This suboption helps to improve the runtime performance of your applications. You can specify a value for dscr depending on your system architecture.

-qsimd=auto

This suboption controls the autosimdization, which was performed by the deprecated **-qhot=simd** option.

-qtune The option default is updated.

The following suboptions are added or updated:

-qtune=pwr7

This suboption specifies that optimizations are tuned for the POWER7 or POWER7+ hardware platforms.

-qtune=pwr8

This suboption specifies that optimizations are tuned for the POWER8 hardware platforms.

SMT suboptions

The new **-qtune** simultaneous multithreading (SMT) suboptions allow you to specify a target SMT to direct optimization for best performance in that mode.

-qunroll=*n*

This suboption hints to the compiler to unroll loops by a factor of *n*. If the loop has fewer than *n* iterations, it is fully unrolled.

-WL This suboption specifies additional options for the IPA link step.

Other XL Fortran updates

This section describes other updates added in AIX, V15.1.

The XLF_POSIX_BINDINGS module

The XLF_POSIX_BINDINGS module provides interfaces to many POSIX and XSI functions and named constants. For details, see The XLF_POSIX_BINDINGS

Enhancements added in Version 14.1

This section describes features and enhancements added to the compiler in Version 14.1. These features and enhancements apply to later versions as well.

Fortran 2008 features

XL Fortran V14.1 implements selected features of the Fortran 2008 standard.

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

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

ALLOCATE enhancements

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

Complex part designators

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

Implied-shape arrays

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

Internal procedures as actual arguments or procedure pointer targets

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

Intrinsic types in the TYPE() type specifier

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

Pointer dummy argument enhancement

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

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

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

The -qx1f2008=checkpresence suboption

The -qx1f2008=checkpresence suboption has been introduced to check the allocation status or pointer association status of actual arguments during argument association of optional dummy arguments.

The BLOCK construct

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

The CONTIGUOUS attribute and IS_CONTIGUOUS intrinsic function

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

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

The END statement for internal and module subprograms

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

The EXECUTE_COMMAND_LINE intrinsic subroutine

The EXECUTE_COMMAND_LINE subroutine has been added in Fortran 2008. You can use it to pass a command to the operating system for execution.

The EXIT statement

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

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

The HYPOT intrinsic procedure

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

The ISO_FORTRAN_ENV intrinsic module

The following constants are added:

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

The following functions are added:

- COMPILER_OPTIONS
- COMPILER_VERSION

The LEADZ and TRAILZ intrinsic procedures

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

The math intrinsic procedures extension

The following new intrinsic procedures have been introduced:

- ACOSH
- ASINH
- ATANH
- ERFC_SCALED
- LOG_GAMMA

Notes:

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

Complex arguments are now supported in the following intrinsic procedures:

- **ACOS**
- **ASIN**
- **ATAN**
- **COSH**
- **SINH**
- **TAN**
- **TANH**

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

The NEWUNIT= specifier

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

The POPCNT and POPPAR inquiry intrinsic functions

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

The RADIX= argument

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

The STOP and ERROR STOP statements

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

OpenMP 3.1

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

OpenMP 3.1 includes the following updates to OpenMP 3.0:

- Adds the **FINAL** and **MERGEABLE** clauses to the **TASK** construct to support optimization.
- Adds the **TASKYIELD** construct to allow users to specify where in the program can perform task switching.
- Adds the `omp_in_final` runtime library routine to support specialization of final task regions.

- Extends the ATOMIC construct to include READ, WRITE, and CAPTURE forms; adds the UPDATE clause to apply the existing form of the ATOMIC construct.
- Allows dummy arguments with the INTENT(IN) attribute to be specified on the FIRSTPRIVATE clause.
- Allows unallocated allocatable arrays to be specified on the COPYIN clause.
- Allows Fortran 90 Pointers to be specified on the FIRSTPRIVATE clause.
- Adds the OMP_PROC_BIND environment variable to control whether OpenMP threads are allowed to move between processors.
- Extends the OMP_NUM_THREADS environment variable to specify the number of threads to use for nested parallel regions.

Related information

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

Performance and optimization

Additional features and enhancements in XL Fortran V14.1 assist with performance tuning and application optimization.

Reports about compiler optimizations

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

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

Diagnostic reports

The new diagnostic reports added in XL Fortran V14.1 can help you identify opportunities to improve the performance of your code.

Compiler reports in HTML format

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

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

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

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

Enhancements to profiling reports

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

Relevance of profiling data

This section shows the relevance of the profiling data to the source code during the **-qpdf1** phase. The relevance is indicated by a number in the range of 0 - 100. The larger the number is, the more relevant the profiling data is to the source code, and the more performance gain can be achieved by using the profiling data.

Missing profiling data

This section might include a warning message about missing profiling data. The warning message is issued for each function for which the compiler does not find profiling data.

Outdated profiling data

This section might include a warning message about outdated profiling data. The compiler issues this warning message for each function that is modified after the **-qpdf1** phase. The warning message is also issued when the optimization level changes from the **-qpdf1** phase to the **-qpdf2** phase.

For detailed information about profile-directed feedback, see "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*.

Enhancements to showpdf reports

In addition to block-counter and call-counter profiling information currently provided, you can also use the **showpdf** utility to view cache-miss profiling and value profiling information. Value profiling and cache-miss profiling information can be displayed only in XML format. However, all the other types of profiling information can be displayed in either text or XML format. In this release, the profile-directed feedback (PDF) information is saved in two files. One is a PDF map file that is generated during the **-qpdf1** phase, and the other is a PDF file that is generated during the execution of the resulting application. You can run the **showpdf** utility to display the PDF information contained in these two files. For more information, see "Viewing profiling information with showpdf" in the *XL Fortran Optimization and Programming Guide*.

New and enhanced diagnostic options

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

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

Table 5. Listings-related compiler options and directives

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

Compiler options and directives

This section describes new or changed compiler options and directives in V14.1.

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

New or changed compiler options

-g, -qdbg

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

-qassert

-qassert=minitercnt=*n* and **-qassert=maxitercnt=*n*** are added to specify the expected minimum and maximum iteration counts of the loops in the program.

-qfunctrace

The **-qfunctrace** option is extended to allow you to specify module procedures and module names.

-qhaltmsg

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

-qinitialloc

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

-qlanglvl

The following suboptions are added or updated:

-qlanglvl=2008std

-qlanglvl=2008pure

These two new suboptions are added to enable language level checking for supported Fortran 2008 features.

-qlistfmt

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

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

-qmaxerr

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

-qoptfile

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

-qpic

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

-qshowpdf

The default value is changed from **-qnoshowpdf** to **-qshowpdf**.

-qx1f2008

The new suboption **-qx1f2008=checkpresence** is added so that you can check dummy argument presence according to the Fortran 2008 standard.

-qx1f2003

The new suboption **-qx1f2003=dynamiacval** is added to control whether you can use unlimited polymorphic entities for array constructors, and whether dynamic types of array constructor values are used.

New or changed directives

ALIGN

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

ASSERT

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

Chapter 5. Setting up and customizing XL Fortran

This section describes how to set up and customize the compiler according to your own requirements.

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

Using custom compiler configuration files

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

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 that you specify in your custom configuration files with compiler settings that are specified in the default configuration file. Compiler updates that might later affect settings in the default configuration file do not affect the settings in your custom configuration files.

Related information



Using custom compiler configuration files

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 6. Developing applications with XL Fortran

Fortran application development consists of repeating cycles of editing, compiling, linking, and running. By default, compiling and linking are combined into a single step.

Notes:

- Before you use the compiler, ensure that XL Fortran is properly installed and configured. For more information, see the *XL Fortran Installation Guide*.
- 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 are executed more than once during a compilation. As each compilation component runs, the results are sent to the next step in the sequence.

1. Preprocessing of source files
2. Compilation, which might 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. Assembling 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 on your system, such as **vi** or **emacs**.

Source programs must be saved using a recognized file name suffix. See “XL Fortran input and output files” on page 50 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*.

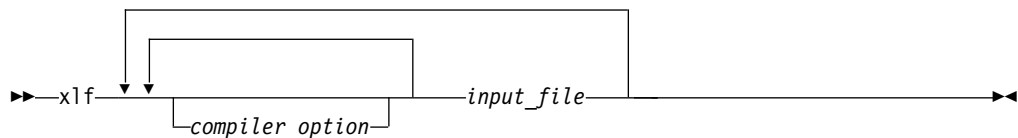
Compiling with XL Fortran

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

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 Fortran source program, use the following basic invocation syntax:



For most applications, compile with `xlf` or a threadsafe counterpart.

- If the file name extensions of your source files indicate a specific level of Fortran, such as `.f08`, `.f03`, `.f95`, `.f90`, or `.f77`, you can compile with `xlf` or the corresponding generic threadsafe invocations so that the compiler can automatically select the appropriate language-level defaults.
- If you compile source files whose file name extensions are generic, such as `.f` or `.F`, with `xlf` or corresponding generic threadsafe invocations, the compilation conforms to FORTRAN 77.

For more information about threadsafe counterparts, see "Compiling XL Fortran programs" in the *XL Fortran Compiler Reference*.

Invocation commands for different levels of Fortran

More invocation commands are available to meet specialized compilation needs, primarily to provide explicit compilation support for different levels and extensions of the Fortran language. These invocation commands do not consider the specific level of Fortran indicated by the source file name extensions, such as `.f08`, `.f03`, `.f95`, `.f90`, or `.f77`.

Table 6. Invocation commands and corresponding Fortran language standards

Language level	Invocation commands	Notes
Fortran 2008	<ul style="list-style-type: none"> • f2008 • xlf2008 	<p>These compiler invocations accept Fortran 90 free source form by default. To use fixed source form with these invocations, you must specify the -qfixed option.</p> <p>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 the I/O formats of Fortran 90 invocations. Switch to the Fortran 95 formats for data files whenever possible.</p>
Fortran 2003	<ul style="list-style-type: none"> • f2003 • xlf2003 	
Fortran 95	<ul style="list-style-type: none"> • f95 • xlf95 	
Fortran 90	<ul style="list-style-type: none"> • f90 • xlf90 	
FORTRAN 77	<ul style="list-style-type: none"> • f77 • fort77 	<p>Where possible, these compiler invocations maintain compatibility with existing programs by using the same I/O formats as FORTRAN 77 and some implementation behaviors that are compatible with earlier versions of XL Fortran.</p> <p>You might need to continue using these invocations for compatibility with existing makefiles and build environments. However, programs that are compiled with these invocations might not conform to the Fortran 2008, Fortran 2003, Fortran 95, or Fortran 90 language level standards.</p>

Compiling with full compliance to language standards

By default, these invocation commands do not conform completely to the corresponding language standards. If you need full compliance, compile with the following compiler option settings and specify the following runtime options before you run the program, with a command similar to the following examples:

Fortran 2008

Compiler options:

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

Example of runtime options:

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

Fortran 2003

Compiler options:

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

Example of runtime options:

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

Fortran 95

Compiler options:

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

Example of runtime options:

```
export XLF RTEOPTS="err_recovery=no:langlvl=95std"
```

Fortran 90

Compiler options:

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

Example of runtime options:

```
export XLF RTEOPTS="err_recovery=no:langlvl=90std"
```

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

The -qxlf2003 compiler option

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

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

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

The -qxlf2008 compiler option

You can use the **-qxlf2008** compiler option for the following purposes:

- To enable language features specific to the Fortran 2008 standard when you compile with compiler invocations that conform to earlier Fortran standards
- To disable language features specific to the Fortran 2008 standard when you compile with compiler invocations that conform to the Fortran 2008 standard

When you compile with the Fortran 2008 compiler invocations, the default setting is **-qxlf2008=checkpresence**. This setting instructs the compiler to check dummy argument presence according to the Fortran 2008 standard.

For all other compiler invocations, the default is **-qxlf2008=nocheckpresence**.

See "Compiling XL Fortran programs" in the *XL Fortran Compiler Reference* for more information about compiler invocation commands available to you.

Compiling parallelized XL Fortran applications

XL Fortran provides threadsafe compiler invocation commands to compile 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 threadsafe components and libraries. The generic XL Fortran threadsafe compiler invocations are as follows:

- xlf_r
- xlf_r7

XL Fortran provides additional threadsafe invocations to meet specific compilation requirements. For more information, see "Compiling XL Fortran programs" in the *XL Fortran Compiler Reference*.

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 the **-qsmp** compiler option. In turn, you should specify the **-qsmp** option only when threadsafe invocations are used. When you specify **-qsmp**, the driver links the libraries that are specified on the `smp libraries` line in the active stanza of the configuration file.

For more information about 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 in one or any combination of the following ways:

- On the command line
- In your source code using directive statements
- In a makefile
- In the stanzas found in a compiler configuration file

You can also pass options to the linker, assembler, and preprocessor.

Priority sequence of compiler options

Option conflicts and incompatibilities might occur when multiple compiler options are specified. To resolve these conflicts in a consistent manner, the compiler applies the following general priority sequence to most options:

1. Directive statements in your source file override command line settings.
2. Compiler option settings on the command line override configuration file settings.
3. Configuration file settings override default settings.

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

Note: Some compiler options, such as the **-I** option, do not follow the priority sequence described above. The compiler searches any directories specified with **-I** in the `xlfcfg` file before it searches the directories specified with **-I** on the command line. The **-I** option is cumulative rather than preemptive.

Related information



Specifying options on the command line

XL Fortran input and output files

The topic describes the file types that 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 7. Input file types

Filename extension	Description
.a	Archive or library files
.f, .F, .f77, .F77, .f90, .F90, .f95, .F95, .f03, .F03, .f08, .F08	Fortran source files
.mod	Module symbol files
.smod 1	Submodule symbol files
.o	Object files
.s	Assembler files
.so	Shared object files
Note: 1	Fortran 2008

Table 8. Output file types

Filename extension	Description
a.out	Default name for executable file created by the compiler
.mod	Module symbol files
.smod 1	Submodule symbol files
.lst	Listing files
.o	Object files
.s	Assembler files

Table 8. Output file types (continued)

Filename extension	Description
.so	Shared object files
Note:  Fortran 2008	

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, you can use `xlf` to compile `file1.f` and `file3.f` to produce object files `file1.o` and `file3.o`; after that, all object files, including `file2.o`, are submitted to the linker to produce one executable.

```
xlf file1.f file2.o file3.f
```

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
```

Linking new objects with existing ones

If you have `.o` or other object files that you compiled with an earlier versions 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

You can use XL Fortran 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 might perform better than statically linked programs if several programs use the same shared routines at the same time. By using dynamic linking, you can upgrade the routines in the shared libraries without relinking. This form of linking is the default and no additional options are needed.

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 run on systems without the XL Fortran runtime libraries. They might 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.

Note: Dynamically and statically linked programs might 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

After a program is compiled and linked, you can run the generated executable file on the command line.

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.

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`.

To run a program, enter the name of the program executable file with runtime arguments on the command line.

Canceling execution

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

To cancel a running program, press **Ctrl+C** 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. Some environment variables do not control actual runtime behavior, but they can have an impact on how your applications run.

For more information about 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 can run with more recent versions of AIX. However, applications linked on a system using a newer version of AIX might 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 need to install a runtime environment on that system or link your application statically.

You can obtain the latest XL Fortran Runtime Environment images, together with licensing and usage information, from the XL Fortran for AIX support page.

XL Fortran compiler diagnostic aids

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

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

- "Understanding XL Fortran compiler listings"
- "Error checking and debugging options"
- "Listings, messages, and compiler information options"

Debugging compiled applications

You can use a symbolic debugger to debug applications compiled with XL Fortran.

At compile time, you can use the **-g** or **-q1 inedebug** option to instruct the XL Fortran compiler to include debugging information in compiled output. For **-g**, you can also use different levels to balance between debug capability and compiler optimization. For more information about the debugging options, see "Error checking and debugging" in the *XL Fortran Compiler Reference*.

You can then use **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 you debug your applications. If you need to debug an optimized application, you can consider using the **-gN** form of the **-g** option along with any optimization options. This form of the **-g** option provides different levels of tradeoff between full optimization and full

debugging support, depending on the value of N. For more information about debugging your optimized code, see "Debugging optimized code" in the *XL Fortran Optimization and Programming Guide*.

Determining which level of XL Fortran is being used

To display the version and release level of XL Fortran that you are using, invoke the compiler with the **-qversion** compiler option.

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

```
xlf -qversion=verbose
```

Notices

Programming interfaces: Intended programming interfaces allow the customer to write programs to obtain the services of IBM XL Fortran for AIX.

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, MD-NC119
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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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

Licensees of this program who want 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:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
U.S.A.

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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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, 2015.

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.

PRIVACY POLICY CONSIDERATIONS:

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, or to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details> in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at <http://www.ibm.com/software/info/product-privacy>.

Trademarks

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

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

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

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

Index

Special characters

- .a files 50
- .f and .F files 50
- .i files 50
- .lst files 50
- .mod files 50
- .o files 50
- .s files 50
- .S files 50

Numerics

- 64-bit environment 5

A

- a.out file 50
- assembler
 - source (.s) files 50
 - source (.S) files 50

B

- backward 17
- backward compatibility issues 17
- basic example, described ix

C

- code optimization 5
- commands 4
- compatibility 17
- compilation
 - sequence of activities 45
- compiler
 - invoking 46
 - running 46
- compiler directives
 - new or changed 40
- compiler options
 - conflicts and incompatibilities 50
 - new or changed 22, 32, 40
 - specification methods 49
- compiling
 - SMP programs 48

D

- dbx debugger 7, 53
- debugger support 53
 - output listings 53
 - symbolic 7
- debugging 53
- debugging compiled applications 53
- debugging information, generating 53
- dynamic linking 52

E

- editing source files 45
- executable files 50
- executing a program 52
- executing the linker 51

F

- f2003 command
 - description 46
 - level of Fortran standard compliance 47
- f2008 command
 - description 46
 - level of Fortran standard compliance 47
- f77 command
 - description 46
 - level of Fortran standard compliance 47
- f90 command
 - description 46
 - level of Fortran standard compliance 47
- f95 command
 - description 46
 - level of Fortran standard compliance 47
- files
 - editing source 45
 - input 50
 - output 50
- fort77 command
 - description 46
 - level of Fortran standard compliance 47
- Fortran 2003
 - compiling programs written for 47
- Fortran 2008
 - compiling programs written for 47
- Fortran 90
 - compiling programs written for 47
- Fortran 95
 - compiling programs written for 47

I

- input files 50
- invocation commands 46
- invoking a program 52
- invoking the compiler 46

L

- language standards 3
- language support 3
- libraries 50
- libxlf.a library 14
- libxlf90.a and libxlf.a libraries 13

- libxlf90.a library 14
- linking
 - dynamic 52
 - static 52
- linking process 51
- listings 50

M

- migrating
 - from previous versions of XL Fortran 13
- migration
 - source code 49
- mod files 50
- more enhancements
 - new features V15.1.3 11
- multiprocessor systems 6

O

- object files 50
 - creating 51
 - linking 51
- OpenMP 6
- optimization
 - programs 5
- output files 50

P

- parallelization 6
- performance
 - optimizing transformations 5
- POSIX Pthreads
 - API support 49
- problem determination 53
- programs
 - running 52

R

- running the compiler 46
- runtime environment 53
- runtime libraries 50
- runtime options 53

S

- shared memory parallelization 6
- shared object files 50
- SMP
 - programs, compiling 48
- SMP programs 6
- source files 50
- source-level debugging support 7
- static linking 52
- symbolic debugger support 7

T

tools

- cleanpdf utility 4
- CreateExportList 4
- custom installation 5
- install 5
- mergepdf utility 4
- showpdf utility 4
- xlfindi 5

U

utilities

- cleanpdf 4
- CreateExportList 4
- custom installation 5
- install 5
- mergepdf 4
- showpdf 4
- xlfindi 5

X

xl.ccfg file 49

xl command

- description 46
- level of Fortran standard compliance 13

xl_r command

- description 46
- for compiling SMP programs 48
- level of Fortran standard compliance 13

xl_r7 command

- description 46
- for compiling SMP programs 48
- level of Fortran standard compliance 13

xl2003 command

- description 46
- level of Fortran standard compliance 47

xl2003_r command

- description 46
- level of Fortran standard compliance 47

xl2008 command

- description 46
- level of Fortran standard compliance 47

xl2008_r command

- description 46
- level of Fortran standard compliance 47

xl90 command

- description 46
- level of Fortran standard compliance 13, 47

xl90_r command

- description 46
- for compiling SMP programs 48
- level of Fortran standard compliance 13, 47

xl90_r7 command

- description 46
- for compiling SMP programs 48

xl90_r7 command (*continued*)

- level of Fortran standard compliance 13, 47

xl95 command

- description 46
- level of Fortran standard compliance 13, 47

xl95_r command

- description 46
- for compiling SMP programs 48
- level of Fortran standard compliance 13, 47

xl95_r7 command

- description 46
- for compiling SMP programs 48
- level of Fortran standard compliance 13, 47



Product Number: 5765-J09; 5725-C74

Printed in USA

SC27-4242-02

