

IBM XL Fortran for Linux, V15.1



# Getting Started with XL Fortran

*Version 15.1*



IBM XL Fortran for Linux, V15.1



# Getting Started with XL Fortran

*Version 15.1*

**Note**

Before using this information and the product it supports, read the information in “Notices” on page 57.

**First edition**

This edition applies to IBM XL Fortran for Linux, V15.1 (Program 5765-J10; 5725-C75) 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, 2014.

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

---

# Contents

<b>About this document . . . . .</b>	<b>v</b>
Conventions . . . . .	v
Related information . . . . .	x
IBM XL Fortran information . . . . .	x
Standards and specifications . . . . .	xi
Other IBM information . . . . .	xii
Technical support . . . . .	xii
How to send your comments . . . . .	xii

<b>Chapter 1. Introducing XL Fortran . . . . .</b>	<b>1</b>
Commonality with other IBM compilers . . . . .	1
Operating system support . . . . .	1
A highly configurable compiler . . . . .	1
Language standard compliance . . . . .	3
Source-code migration and conformance checking . . . . .	3
Tools, utilities, and commands . . . . .	3
Program optimization . . . . .	5
64-bit object capability . . . . .	5
Shared memory parallelization . . . . .	6
Diagnostic listings . . . . .	6
Symbolic debugger support . . . . .	7

<b>Chapter 2. What's new for IBM XL Fortran for Linux, V15.1 . . . . .</b>	<b>9</b>
Support for POWER8 processors. . . . .	9
Fortran 2008 features . . . . .	10
Language interoperability features. . . . .	12
OpenMP 4.0 . . . . .	13
Directives and intrinsic procedures . . . . .	14
Compiler options . . . . .	16
Other XL Fortran updates . . . . .	18

<b>Chapter 3. Migrating from earlier versions . . . . .</b>	<b>21</b>
Compatibility with earlier versions . . . . .	21
Enhancements added in Version 14.1 . . . . .	24
Fortran 2008 features . . . . .	24
OpenMP 3.1 . . . . .	27
Performance and optimization . . . . .	28
Diagnostic reports . . . . .	28
Compiler options and directives . . . . .	30
Enhancements added in Version 13.1 . . . . .	31

Support for POWER7 processors . . . . .	31
XL Fortran language-related updates . . . . .	32
OpenMP 3.0 . . . . .	33
Performance and optimization . . . . .	34
New diagnostic reports . . . . .	35
Utilization tracking and reporting tool . . . . .	37
New or changed compiler options and directives . . . . .	38
Directives and intrinsics . . . . .	39
Enhancements added in Version 12.1 . . . . .	40
XL Fortran language-related updates . . . . .	40
OpenMP 3.0 . . . . .	41
Performance and optimization . . . . .	41
Compiler options and directives . . . . .	42

<b>Chapter 4. Setting up and customizing XL Fortran . . . . .</b>	<b>45</b>
Using custom compiler configuration files . . . . .	45
Configuring compiler utilization tracking and reporting . . . . .	45

<b>Chapter 5. Developing applications with XL Fortran . . . . .</b>	<b>47</b>
The compiler phases . . . . .	47
Editing Fortran source files . . . . .	47
Compiling with XL Fortran . . . . .	48
Invoking the compiler . . . . .	48
Compiling parallelized XL Fortran applications . . . . .	50
Specifying compiler options . . . . .	51
XL Fortran input and output files . . . . .	52
Linking your compiled applications with XL Fortran . . . . .	52
Dynamic and static linking . . . . .	53
Running your compiled application . . . . .	53
XL Fortran compiler diagnostic aids . . . . .	54
Debugging compiled applications . . . . .	54
Determining which level of XL Fortran is being used. . . . .	55

<b>Notices . . . . .</b>	<b>57</b>
Trademarks and service marks . . . . .	59

<b>Index . . . . .</b>	<b>61</b>
------------------------	-----------



---

## About this document

This document contains overview and basic usage information for the IBM® XL Fortran for Linux, V15.1 compiler.

### Who should read this document

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

### How to use this document

Throughout this document, the `xlf` compiler invocation is used to describe the actions of the compiler. You can, however, substitute other forms of the compiler invocation command if your particular environment requires it, and compiler option usage will remain the same unless otherwise specified.

While this document covers information on configuring the compiler environment, and compiling and linking Fortran applications using the XL Fortran compiler, it does not include the following topics:

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

---

## Conventions

### Typographical conventions

The following table shows the typographical conventions used in the IBM XL Fortran for Linux, V15.1 information.

Table 1. *Typographical conventions*

Typeface	Indicates	Example
<b>lowercase bold</b>	Invocation commands, executable names, and compiler options.	The compiler provides basic invocation commands, <b>xlf</b> , 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 <b>a.out</b> .
<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.
<b>UPPERCASE bold</b>	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 <b>ASSERT</b> directive applies only to the <b>DO</b> loop immediately following the directive, and not to any nested <b>DO</b> 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
 <b>F2008</b>  <b>F2008</b>	N/A	The text describes an IBM XL Fortran implementation of the Fortran 2008 standard.
 <b>F2003</b>  <b>F2003</b>	Fortran 2003 begins / ends	The text describes an IBM XL Fortran implementation of the Fortran 2003 standard, and it applies to all later standards.
 <b>IBM</b>  <b>IBM</b>	IBM extension begins / ends	The text describes a feature that is an IBM XL Fortran extension to the standard language specifications.

Table 2. Qualifying elements (continued)

Icon	Bracket separator text	Meaning
<div>► TS</div> <div>TS ◄</div>	N/A	The text describes a feature in a Technical Specification that is not part of the current Fortran standard.

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

## Syntax diagrams

Throughout this information, diagrams illustrate XL Fortran syntax. This section will help you to interpret and use those diagrams.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.  
 The ►— symbol indicates the beginning of a command, directive, or statement.  
 The —► symbol indicates that the command, directive, or statement syntax is continued on the next line.  
 The ►— symbol indicates that a command, directive, or statement is continued from the previous line.  
 The —►◄ symbol indicates the end of a command, directive, or statement.  
 Fragments, which are diagrams of syntactical units other than complete commands, directives, or statements, start with the |— symbol and end with the —| symbol.  
 IBM XL Fortran extensions are marked by a number in the syntax diagram with an explanatory note immediately following the diagram.  
 Program units, procedures, constructs, interface blocks and derived-type definitions consist of several individual statements. For such items, a box encloses the syntax representation, and individual syntax diagrams show the required order for the equivalent Fortran statements.
- Required items are shown on the horizontal line (the main path):

►—keyword—*required\_argument*—►◄

- Optional items are shown below the main path:

►—keyword—  
                   └*optional\_argument*┘—►◄

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

- If you can choose from two or more items, they are shown vertically, in a stack.  
 If you *must* choose one of the items, one item of the stack is shown on the main path.

►—keyword—  
                   └*required\_argument1*  
                   └*required\_argument2*┘—►◄

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



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



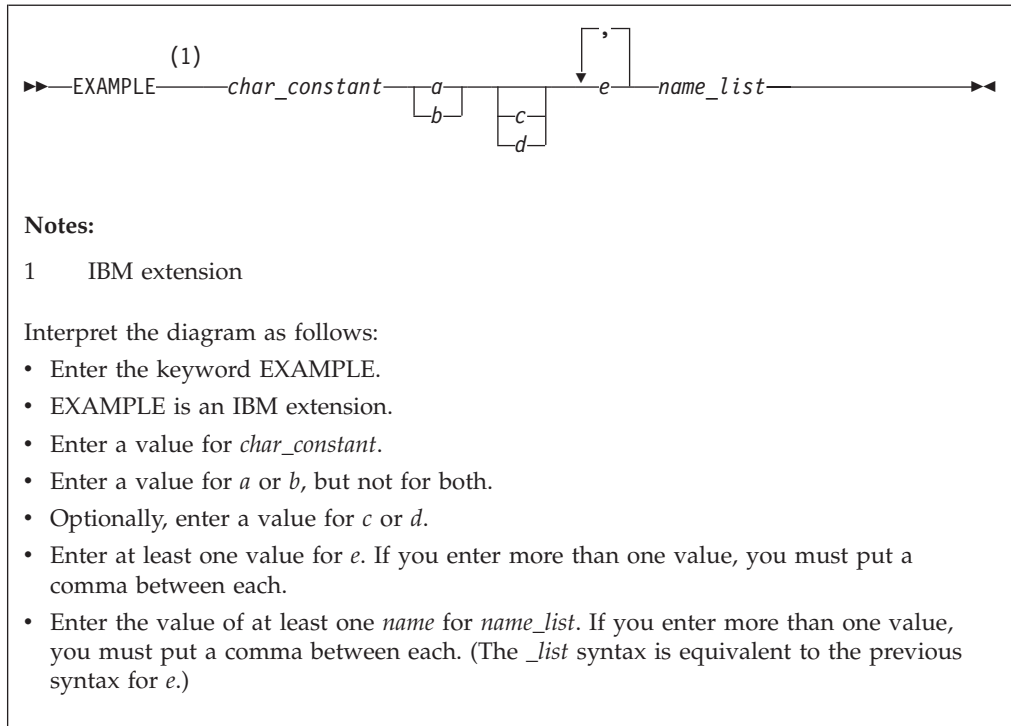
- The item that is the default is shown above the main path.



- Keywords are shown in nonitalic letters and should be entered exactly as shown.
- Variables are shown in italicized lowercase letters. They represent user-supplied names or values. If a variable or user-specified name ends in *\_list*, you can provide a list of these terms separated by commas.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

### Sample syntax diagram

The following is an example of a syntax diagram with an interpretation:



## How to read syntax statements

Syntax statements are read from left to right:

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

### Example of a syntax statement

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

The following list explains the syntax statement:

- Enter the keyword `EXAMPLE`.
- Enter a value for `char_constant`.
- Enter a value for `a` or `b`, but not for both.
- Optionally, enter a value for `c` or `d`.
- Enter at least one value for `e`. If you enter more than one value, you must put a comma between each.
- Optionally, enter the value of at least one `name` for `name_list`. If you enter more than one value, you must put a comma between each `name`.

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

## Examples in this information

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

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

## Notes on the terminology used

Some of the terminology in this information is shortened as follows:

- The term *free source form format* often appears as *free source form*.
- The term *fixed source form format* often appears as *fixed source form*.
- The term *XL Fortran* often appears as *XLF*.

---

## Related information

The following sections provide related information for XL Fortran:

### IBM XL Fortran information

XL Fortran provides product information in the following formats:

- README files  
README files contain late-breaking information, including changes and corrections to the product information. README files are located by default in the XL Fortran directory and in the root directory of the installation CD.
- Installable man pages  
Man pages are provided for the compiler invocations and all command-line utilities provided with the product. Instructions for installing and accessing the man pages are provided in the *IBM XL Fortran for Linux, V15.1 Installation Guide*.
- Information center  
The fully searchable HTML-based documentation is viewable on the web at [http://www.ibm.com/support/knowledgecenter/SSAT4T\\_15.1.0/com.ibm.compilers.linux.doc/welcome.html](http://www.ibm.com/support/knowledgecenter/SSAT4T_15.1.0/com.ibm.compilers.linux.doc/welcome.html).
- PDF documents  
PDF documents are located by default in the `/opt/ibm/xlf/15.1.0/doc/LANG/pdf/` directory, where *LANG* is one of *en\_US* or *ja\_JP*. The PDF files are also available on the web at <http://www.ibm.com/support/docview.wss?uid=swg27036672>.

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 Linux, V15.1 Installation Guide, SC27-4253-00</i>	install.pdf	Contains information for installing XL Fortran and configuring your environment for basic compilation and program execution.

Table 3. XL Fortran PDF files (continued)

Document title	PDF file name	Description
<i>Getting Started with IBM XL Fortran for Linux, V15.1, SC27-4252-00</i>	getstart.pdf	Contains an introduction to the XL Fortran product, with information on setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
<i>IBM XL Fortran for Linux, V15.1 Compiler Reference, SC27-4254-00</i>	compiler.pdf	Contains information about the various compiler options and environment variables.
<i>IBM XL Fortran for Linux, V15.1 Language Reference, SC27-4255-00</i>	langref.pdf	Contains information about the Fortran programming language as supported by IBM, including language extensions for portability and conformance to nonproprietary standards, compiler directives and intrinsic procedures.
<i>IBM XL Fortran for Linux, V15.1 Optimization and Programming Guide, SC27-4256-00</i>	proguide.pdf	Contains information on advanced programming topics, such as application porting, interlanguage calls, floating-point operations, input/output, application optimization and parallelization, and the XL Fortran high-performance libraries.

To read a PDF file, use 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, tutorials, documentation errata, and other articles, is available on the web at:

<http://www.ibm.com/support/docview.wss?uid=swg27036672>

**Note:** Documentation errata is reflected only in the English version of the information center.

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 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.)
- *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 4.0 (Partial support)*, available at <http://www.openmp.org>

## Other IBM information

- *ESSL for AIX V5.1/ESSL for Linux on POWER® V5.1 Guide and Reference* available at the Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL web page.

---

## Technical support

Additional technical support is available from the XL Fortran Support page at [http://www.ibm.com/support/entry/portal/overview/software/rational/xl\\_fortran\\_for\\_linux](http://www.ibm.com/support/entry/portal/overview/software/rational/xl_fortran_for_linux). This page provides a portal with search capabilities to a large selection of Technotes and other support information.

If you cannot find what you need, you can send email to [compinfo@ca.ibm.com](mailto:compinfo@ca.ibm.com).

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

---

## How to send your comments

Your feedback is important in helping to provide accurate and high-quality information. If you have any comments about this information or any other XL Fortran information, send your comments by email to [compinfo@ca.ibm.com](mailto:compinfo@ca.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 Linux, V15.1 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 Linux, V15.1 is part of a larger family of IBM C, C++, and Fortran compilers. XL Fortran, together with 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®/P, IBM Blue Gene®/Q, IBM i, selected Linux distributions, IBM z/OS®, and IBM z/VM®. The common code base, along with compliance with international programming language standards, helps support consistent compiler performance and ease of program portability across multiple operating systems and hardware platforms.

---

### Operating system support

This section describes the operating systems that IBM XL Fortran for Linux, V15.1 supports.

IBM XL Fortran for Linux, V15.1 supports the following operating systems:

- SUSE Linux Enterprise Server 11 Service Pack 2 (SLES 11 SP2) or later
- Red Hat Enterprise Linux 6.4 (RHEL 6.4) or later
- Red Hat Enterprise Linux 7.0 (RHEL 7.0) or later

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

IBM XL Fortran for Linux, V15.1 is supported on big-endian systems. 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`. Each invocation command is unique in that it instructs the compiler to tailor compilation output to meet a specific language level specification. Compiler invocation commands are provided to support all standardized Fortran language levels, and many popular language extensions as well.

The compiler also provides corresponding "`_r`" versions of most invocation commands, for example, `xl_r`. The "`_r`" invocations instruct the compiler to link and bind object files to thread safe components and libraries, and produce thread safe object code for compiler-created data and procedures.

For more information about XL Fortran compiler invocation commands, see "Compiling XL Fortran programs" in the *XL Fortran Compiler Reference*.

## Compiler options

You can choose from a large selection of compiler options to control compiler behavior. You can benefit from using different options for the following tasks:

- Debugging your applications
- Optimizing and tune 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 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 45.

## 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

IBM XL Fortran for Linux, V15.1 supports the following Fortran programming language specifications.

- 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`, then use the generic compiler invocations such as `xl f` or `xl f_r` to automatically select the appropriate language level appropriate to the filename extension.

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

---

## Tools, utilities, and commands

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

## 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

### new\_install

The **new\_install** utility configures IBM XL Fortran for Linux, V15.1 for use on your system after you install the compiler.

### xlf\_configure

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

## Commands

### genhtml command

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

### 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.

#### resetpdf command

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

#### 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*.

---

## 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
- Improving the usage of the memory subsystem
- Exploiting the ability of the architecture to handle large amounts of shared memory parallelization

For more information, see these related topics:

- "Optimizing your applications" in the *XL Fortran Optimization and Programming Guide*
- "Optimizing and tuning options" in the *XL Fortran Compiler Reference*

---

## 64-bit object capability

The XL Fortran compiler's 64-bit object capability addresses increasing demand for larger storage requirements and greater processing power.

The Linux operating system provides an environment that allows you to develop and execute programs that exploit 64-bit processors through the use of 64-bit address spaces.

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

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

XL 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
- Instructing the compiler to automatically generate shared memory parallelization
- Message passing based shared or distributed memory parallelization (MPI)

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

### OpenMP directives

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

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

OpenMP directives address three important issues of parallelizing an application:

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

As of IBM XL Fortran for Linux, V15.1, XL Fortran supports OpenMP API Version 3.1 and selected features of the OpenMP API Version 4.0 specification. For details, see "OpenMP 4.0" on page 13.

For more information about program performance optimization, see:

- "Optimizing your applications" in the *XL Fortran Optimization and Programming Guide*
- The OpenMP API specification for parallel programming

---

## Diagnostic listings

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

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

You can also get the diagnostic information from the compiler in XML or HTML format about some of the optimizations that the compiler performed or missed. 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.

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

The debugging information can be examined by **gdb** or any other symbolic debugger to help you debug your programs.



---

## Chapter 2. What's new for IBM XL Fortran for Linux, V15.1

This section describes features and enhancements added to IBM XL Fortran for Linux, V15.1.

---

### Support for POWER8 processors

XL Fortran for Linux, 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:

- Assumed-rank objects
- Assumed-type objects
- 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

## Assumed-rank objects

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

## Assumed-type objects

Assumed-type objects 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*.

---

## OpenMP 4.0

IBM XL Fortran for Linux, 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

### 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.

### 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

**Note:** POWER8 directives and intrinsic procedures are valid only when **-qarch** is set to target POWER8 processors.

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*.

---

## 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 **pwr5**. 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.
- qdbgfmt** The following suboption is added:
  - qdbgfmt=dwarf** This suboption generates debugging information in DWARF 3 format.
  - qdbgfmt=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.

**-qstaticlink=xlibs**

This suboption links in XL compiler libraries statically.

**-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 IBM XL Fortran for Linux, 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 module in the *XL Fortran Language Reference*.



---

## Chapter 3. Migrating from earlier versions

By migrating to the latest version of the compiler, you are able to take advantage of the new features of the compiler, including features to help boost the performance of the applications being compiled.

A later version of the compiler includes new and enhanced features. These features can provide the following benefits:

- Improved optimization of compiled applications with additional performance benefits
- Support for new language standards to facilitate code portability between multiple operating systems and hardware platforms
- Exploitation of the latest functionality in new hardware and operating systems

Migrating to the latest version of the compiler gives you the ability to exploit the new features with benefits of increased performance optimization, support of new language specifications, and exploitation of new hardware and software environments.

For detailed information about the new features in the current release, see Chapter 2, “What's new for IBM XL Fortran for Linux, V15.1,” on page 9. The white paper *Upgrading XL Fortran Compilers* also provides additional information for compiler migration.

In general, a new version of the compiler is compatible with its earlier versions. However, there could be exceptions. For example, different diagnostic messages could be generated. Always back up your source code and other important data before migrating the compiler.

The following sections list the enhancements added to the compiler in earlier versions, which can assist you in migrating to a later version of the compiler.

---

### 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 command `xlf_r` no longer lists `-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`, 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` no longer implies `-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**, 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 **readelf -s** command to determine whether you need to use the **-qsmp=noostls** suboption. The following code is an example that shows how to use the **readelf -s** command:

```
> readelf -s 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 :: 1   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 =&gt; 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 :: 1     integer a(1)   contains     procedure :: bar =&gt; 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.

---

## Enhancements added in Version 14.1

This section describes features and enhancements added to the compiler in Version 14.1.

### 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 `-qxlf2008=checkpresence` suboption**

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

### **The `BLOCK` construct**

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

### **The `CONTIGUOUS` attribute and `IS_CONTIGUOUS` intrinsic function**

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

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

### **The `END` statement for internal and module subprograms**

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

## The 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](http://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 4. Listings-related compiler options and directives

Option/directive	Description
<b>-qlistfmt</b>	<p>The <b>-qlistfmt</b> 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.

#### **-qhaltonmsg**

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

#### **-qinitalloc**

The new option **-qinitalloc** 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.

**-qpvc**

**-qpvc=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**.

**-qxf2008**

The new suboption **-qxf2008=checkpresence** is added so that you can check dummy argument presence according to the Fortran 2008 standard.

**-qxf2003**

The new suboption **-qxf2003=dynamicacval** is added to control whether you can use unlimited polymorphic entities for array constructors, and whether dynamic types of array constructor values are used.

**New or changed directives****ALIGN**

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

**ASSERT**

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

---

**Enhancements added in Version 13.1**

This section describes features and enhancements added to the compiler in Version 13.1.

**Support for POWER7 processors**

XL Fortran for Linux, V13.1 supports POWER7 processors.

The new features and enhancements introduced in support for the POWER7 processors, fall under the following four categories:

- Vector scalar extension data types and intrinsic procedures
- MASS libraries for POWER7 processors
- Hardware directives and intrinsics for POWER7 processors
- Compiler options for POWER7 processors

**Vector scalar extension data types and intrinsic procedures**

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

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

## Mathematical Acceleration Subsystem (MASS) libraries for POWER7 processors

### Vector libraries

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

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

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

- exp2
- exp2m1
- log21p
- log2

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

### SIMD libraries

The MASS SIMD library **libmass\_simdp7.a** contains an accelerated set of frequently used math intrinsic procedures that provide improved performance over the corresponding standard system library procedures.

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

## Hardware directives and intrinsics for POWER7 processors

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

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

For more information, see “Directives and intrinsics” on page 39.

## New compiler options for POWER7 processors

### New arch and tune compiler options

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

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

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

## XL Fortran language-related updates

XL Fortran V13.1 fully implements the Fortran 2003 standard.

## Fortran 2003 compliance

XL Fortran V13.1 provides the following features:

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

For more information, see Fortran 2003.

## OpenMP 3.0

In V13.1, XL Fortran fully implements the OpenMP API Version 3.0 specification. The XL Fortran implementation is based on IBM's interpretation of the OpenMP Application Program Interface Version 3.0.

Features implemented for OpenMP in V13.1 are:

- Full support for OpenMP task level parallelizations. The new OpenMP constructs TASK and TASKWAIT give users the ability to parallelize irregular algorithms, such as pointer chasing or recursive algorithms for which the existing OpenMP constructs were not adequate.
- Allocatable arrays. You can specify allocatable arrays on PRIVATE, FIRSTPRIVATE, LASTPRIVATE, REDUCTION, COPYIN, and COPYPRIVATE clauses.
- Nested parallelism. A list of runtime routines are available for you to set or get the nested levels and thread limit. These include:
  - omp\_get\_thread\_limit
  - omp\_get\_max\_active\_levels
  - omp\_set\_max\_active\_levels

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

- omp\_get\_level
  - omp\_get\_ancestor\_thread\_num
  - omp\_get\_team\_size
  - omp\_get\_active\_level
- Stack size control. You can now control the size of the stack for threads created by the OpenMP runtime library using the new environment variable OMP\_STACKSIZE.
- Users can give hints to the expected behavior of waiting threads using the new environment variable OMP\_WAIT\_POLICY.
- The storage of a private variable is not allowed to reuse the storage of the original variable on the master thread.
- Some restrictions on the PRIVATE clause have been removed. A variable that appears in the REDUCTION clause of a parallel construct can now also appear in a PRIVATE clause on a work-sharing construct.
- A new SCHEDULE type, AUTO, allows the compiler and runtime system to control scheduling. You can use the following routines to get or set schedule type:
  - omp\_get\_schedule
  - omp\_set\_schedule
- Consecutive loop constructs with STATIC schedule with NOWAIT clause now guarantee the same iterations are being assigned to the same thread in the constructs.

- You can set the `OMP_THREAD_LIMIT` environment variable to determine the number of OpenMP threads to use for the whole program. Set `OMP_MAX_ACTIVE_LEVELS` if you want to control the maximum number of nested, active parallel regions.

The following is an enhancement to OpenMP in this release:

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

For more information, see:

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

## Performance and optimization

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

### Enhancements to `-qpdf`

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

In previous releases, if you modify the source files and compile them with the `-qpdf2` option, the compilation stops with an error. As of XL Fortran for Linux, V13.1, the compiler issues a list of warnings but the compilation does not stop. This allows you to continue using the profiling data after modifying the source files.

Some new suboptions are added to the `-qpdf` option. You can use these new suboptions to get more control over performance improvements and extend `-qpdf` to support multiple-pass profiling, cache-miss profiling, and extended value profiling.

The new `-qpdf` suboptions are:

**level** Supports multiple-pass profiling, single-pass profiling, cache-miss profiling, value profiling, block-counter profiling, and call-counter profiling. You can compile your program with `-qpdf1=level=0|1|2` to specify the type of profiling information to be generated by the resulting application.

**exename**

Specifies the name of the generated PDF file according to the output file name specified by the `-o` option.

**defname**

Reverts the PDF file to its default file name.

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

## Reports about compiler optimizations

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

## Performance-related compiler options and directives

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

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

Table 5. Performance-related compiler options and directives

<b>-qinline=level=number</b>	A new option is added to <b>-qinline</b> to provide guidance to the compiler about the relative value of inlining in relation to the default value of 5. <i>number</i> is a range of integer values 0 - 10 that indicates the level of inlining you want to use. For details, see <b>-qinline</b> in the <i>XL Fortran Compiler Reference</i> .
<b>-qpdf</b>	<b>-qpdf</b> provides suboptions to give you more control flexibility in controlling different PDF optimizations. For more information, see the <b>-qpdf1</b> , <b>-qpdf2</b> section in the <i>XL Fortran Compiler Reference</i> .
<b>-qprefetch</b>	A new enhancement is added to <b>-qprefetch</b> for inserting prefetch instructions automatically where there are opportunities to improve code performance: <b>-qprefetch=assistthread</b> . For details, see <b>-qprefetch</b> in the <i>XL Fortran Compiler Reference</i> .

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

## New diagnostic reports

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

## Compiler reports in XML format

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

In this release, the following four optimization categories are available in the report:

- Inlining
- Loop transformations
- Data reorganizations
- Profile-directed feedback information

The new **-qlistfmt** option and its associated suboptions can be used to generate the new XML 1.0 report.

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

## Enhancements to profiling reports

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:

### Loop iteration count

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

### Block and call count

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

### Cache miss

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

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

For detailed information about 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*.

## Report of data reorganization

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

- Data reorganizations (a summary of how program variable data gets reorganized by the compiler)

- The location of data prefetch instructions inserted by the compiler

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

Reorganizations include:

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

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

## Additional loop analysis

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

## New and enhanced diagnostic options

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

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 6. Listings-related compiler options and directives

Option/directive	Description
<b>-qlistfmt</b>	Generates a report in an XML 1.0 format containing information about optimizations performed by the compiler and missed optimization opportunities. The report contains information about inlining, loop transformations, data reorganization and profile-directed feedback.
<b>-qreport</b>	The listing now contains a PDF report section when used with <b>-qpdf2</b> . Another new section in the listing files is a DATA REORGANIZATION section when used with <b>-qipa=level=2</b> or <b>-O5</b> .

## Utilization tracking and reporting tool

The utilization tracking and reporting feature is a lightweight and simple mechanism for tracking the compiler utilization within your organization. It is

disabled by default. You can use this feature to detect whether your organization's use of the compiler exceeds your compiler license entitlements.

When utilization tracking is enabled, each invocation of the compiler is recorded in a compiler utilization file. You can run the utilization reporting tool to generate a report from one or more of these files to get a picture of the overall usage of the compiler within your organization. The **urt** command can be used to control how the report is generated. In particular, the report indicates the number of concurrent users using the compiler.

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

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

## New or changed compiler options and directives

This section describes new and changed compiler options and directives in XL Fortran, V13.1.

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.

Table 7. New or changed compiler options and directives

Option or directive	Description
<b>-qarch</b>	A new suboption has been added to <b>-qarch</b> , specifying <b>-qarch=pwr7</b> produces object code that contains instructions that run on the POWER7 hardware platforms.
<b>-qassert</b>	New suboptions have been added for <b>-qassert</b> to provide more information about the characteristics of the files that can help to fine-tune optimizations.  The <b>-qassert=contig</b> option tells the compiler that all array pointers are associated with contiguous targets, and that all assumed-shape arrays are associated with contiguous actual arguments.  The <b>-qassert=refalign</b> option tells the compiler that all pointers only point to naturally-aligned data.
<b>-qbindcextname</b>	Controls whether the <b>-qextname</b> option affects <b>BIND(C)</b> entities.
<b>-qfunctrace</b>	Inserts calls to user-defined tracing procedures at procedure entry and exit.
<b>-qfunctrace_xlf_catch</b>	Specifies the name of the catch tracing subroutine.
<b>-qfunctrace_xlf_enter</b>	Specifies the name of the entry tracing subroutine.
<b>-qfunctrace_xlf_exit</b>	Specifies the name of the exit tracing subroutine.
<b>-qinline</b>	Attempts to inline functions instead of generating calls to those functions, for improved performance.
<b>-qlibmpi</b>	Tunes code based on the known behavior of the Message Passing Interface (MPI) functions.

Table 7. New or changed compiler options and directives (continued)

Option or directive	Description
<b>-qlistfmt</b>	Generates a report in an XML 1.0 format containing information about some optimizations performed by the compiler and some missed optimization opportunities for inlining, loop transformations, profile-directed feedback, and data reorganization.
<b>-qmkshrobj</b>	Creates a shared object from generated object files.
<b>-qstackprotect</b>	Protects your applications against malicious code or programming errors that overwrite or corrupt the stack.
<b>-qstaticlink</b>	Controls how shared and nonshared runtime libraries are linked into an application.
<b>-qstrict</b>	A new suboption has been added to the <b>-qstrict</b> option to allow more control over optimizations and transformations that violate strict program semantics.  <b>-qstrict=vectorprecision</b> disables vectorization in loops where it might produce different results in vectorized iterations than in nonvectorized ones.
<b>-qtune</b>	A new suboption has been added to <b>-qtune</b> . If you specify <b>-qtune=pwr7</b> , optimizations are tuned for the POWER7 hardware platforms.

Table 8. Deprecated directives and options

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

## Directives and intrinsics

This section lists directives and intrinsics that are new for XL Fortran, V13.1.

## VSX built-in functions

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

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

## POWER7 prefetch extensions and cache control

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

- PROTECTED\_STREAM\_STRIDE
- TRANSIENT\_PROTECTED\_STREAM\_COUNT\_DEPTH
- UNLIMITED\_PROTECTED\_STREAM\_DEPTH
- TRANSIENT\_UNLIMITED\_PROTECTED\_STREAM\_DEPTH
- PARTIAL\_DCBT
- DCBTT
- DCBTSTT
- DCBFLP

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

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

## POWER7 intrinsics

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

- BPERMD
- DIVDE
- DIVWE

## Comparison intrinsics

This new function compares bytes.

- CMPB

---

## Enhancements added in Version 12.1

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

## XL Fortran language-related updates

This section describes the language-related changes introduced in Version 12.1.

## Fortran 2003 enhancements

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

## IEEE module enhancements

The IEEE\_ARITHMETIC module defines a new constant, IEEE\_OTHER\_VALUE.

The IEEE\_ARITHMETIC module defines three new functions:  
IEEE\_SET\_UNDERFLOW\_MODE, IEEE\_GET\_UNDERFLOW\_MODE and  
IEEE\_SUPPORT\_UNDERFLOW\_MODE.

## OpenMP 3.0

IBM XL Fortran for Linux, V12.1, has added some of the features of the OpenMP API Version 3.0 specification. The XL Fortran implementation is based on IBM's interpretation of the OpenMP Application Program Interface Draft 3.0 Public Comment.

The main differences between Version 2.5 and Version 3.0 implemented in this release are:

- Addition of task level parallelization. The new OpenMP constructs TASK and TASKWAIT give users the ability to parallelize irregular algorithms, such as pointer chasing or recursive algorithms for which the existing OpenMP constructs were not adequate.
- Nesting support - a COLLAPSE clause has been added to the DO, and PARALLEL DO directives to allow parallelization of perfect loop nests. This means that multiple loops in a nest can be parallelized.

For more information, see:

- "Parallel programming with XL Fortran" in the *XL Fortran Optimization and Programming Guide*
- [www.openmp.org](http://www.openmp.org)

## Performance and optimization

XL Fortran, V12.1 includes features and enhancements to assist with performance tuning and optimization of your applications.

### Enhancements to -qstrict

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

You can use 16 new suboptions separately or use a suboption group. Here is a list of suboption groups:

- all** Disables all semantics-changing transformations, including those controlled by the other suboptions.
- ieeefp** Controls whether individual operations conform to IEEE 754 semantics.
- order** Controls whether individual operations can be reordered in a way that violate program language semantics.
- precision** Controls optimizations and transformations that can affect the precision of program results.
- exceptions** Controls optimizations and transformations that can affect the runtime exceptions generated by the program.

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

## Performance-related compiler options and directives

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

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

*Table 9. Performance-related compiler options and directives*

Option/directive	Description
<b>-qfloat</b>	Some <b>-qfloat</b> suboptions are affected by the new suboptions for <b>-qstrict</b> .
<b>EXECUTION_FREQUENCY</b>	The <b>EXECUTION_FREQUENCY</b> directive marks source code that might be executed very frequently or very infrequently. When optimization is enabled, the directive is used as a hint to the optimizer.
<b>-qreport</b>	The listing now contains information about how many streams are created for each loop and which loops cannot be SIMD vectorized due to non-stride-one references. You can use this information to improve the performance of your applications.
<b>-qsmp</b>	When <b>-qsmp=omp</b> is in effect, some of the additional functionality of OpenMP API 3.0 is now available. For more information, see "OpenMP 3.0" on page 33.

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

## Compiler options and directives

This section describes new and changed compiler options and directives in XL Fortran, V12.1.

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.

*Table 10. New or changed compiler options and directives*

Option or directive	Description
<b>-qstrict</b>	Many suboptions have been added to the <b>-qstrict</b> option to allow more control over optimizations and transformations that violate strict program semantics. See “Performance and optimization” on page 34 for more information.
<b>EXECUTION_FREQUENCY</b>	The <b>EXECUTION_FREQUENCY</b> directive marks source code that you expect will be executed very frequently or very infrequently.
<b>IGNORE_TKR</b>	The <b>IGNORE_TKR</b> directive facilitates portability of code written for other compilers. It simplifies the writing of generic interfaces, especially for system libraries by directing the compiler to ignore type, kind and rank of dummy arguments.
<b>-qreport</b>	When used together with compiler options that enable automatic parallelization or vectorization, the <b>-qreport</b> option now reports the number of streams in a loop and produces information when loops cannot be SIMD vectorized due to non-stride-one references.
<b>-qsmp</b>	When <b>-qsmp=omp</b> is in effect, some of the additional functionality of OpenMP API 3.0 is now available. For more information, see “OpenMP 3.0” on page 41.
<b>-qtimestamps</b>	This option can be used to remove timestamps from generated binaries.



---

## Chapter 4. Setting up and customizing XL Fortran

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 does not affect the settings in your custom configuration files.

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

---

### Configuring compiler utilization tracking and reporting

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

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

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



---

## Chapter 5. Developing applications with XL Fortran

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

### Notes:

1. Before you can use the compiler, you must first ensure that XL Fortran is properly installed and configured. For more information, see the *XL Fortran Installation Guide*.
2. To learn about writing Fortran programs, refer to the *XL Fortran Language Reference*.

---

### The compiler phases

A typical compiler invocation executes some or all of these activities in sequence. For link time optimizations, some activities 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 may consist of the following phases, depending on what compiler options are specified:
  - a. Front-end parsing and semantic analysis
  - b. Loop transformations
  - c. High-level optimization
  - d. Low-level optimization
  - e. Register allocation
  - f. Final assembly
3. Assemble the assembly (.s) files, and the unpreprocessed assembler (.S) files after they are preprocessed
4. Object linking to create an executable application

To see the compiler step through these phases, specify the **-v** compiler option when you compile your application. To see the amount of time the compiler spends in each phase, specify **-qphsinfo**.

---

### Editing Fortran source files

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

Source programs must be saved using a recognized file name suffix. See “XL Fortran input and output files” on page 52 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*.

---

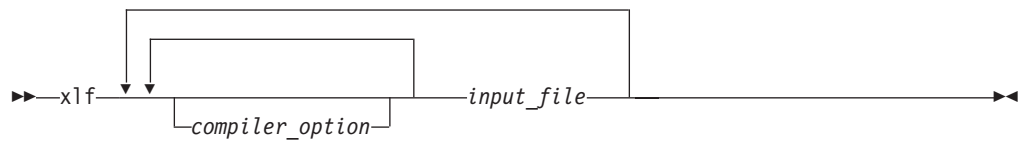
## Compiling with XL Fortran

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 corresponding generic threadsafe invocations so 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 11. Invocation commands and corresponding Fortran language standards

Language level	Invocation commands	Notes	
Fortran 2008	<ul style="list-style-type: none"><li>• f2008</li><li>• xlf2008</li></ul>	These compiler invocations accept Fortran 90 free source form by default. To use fixed source form with these invocations, you must specify the <b>-qfixed</b> option.	The Fortran 2008 language standard is partially supported in this release.
Fortran 2003	<ul style="list-style-type: none"><li>• f2003</li><li>• xlf2003</li></ul>		
Fortran 95	<ul style="list-style-type: none"><li>• f95</li><li>• xlf95</li></ul>		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.
Fortran 90	<ul style="list-style-type: none"><li>• f90</li><li>• xlf90</li></ul>		
FORTRAN 77	<ul style="list-style-type: none"><li>• f77</li><li>• fort77</li></ul>	Where possible, these compiler invocations maintain compatibility with existing programs by using the same I/O formats as FORTRAN 77 and some implementation behaviors compatible with earlier versions of XL Fortran.  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.	

## 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:nofold -qnoswapomp -qstrictieeeemod
```

#### Example of runtime options:

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

### Fortran 2003

#### Compiler options:

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

#### Example of runtime options:

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

### Fortran 95

**Compiler options:**

```
-qlanglvl=95std -qnodirective -qnoescape -qextname
-qfloat=nomaf: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: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 invocation is as follow:

- xlf\_r

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 OpenMP directives and activate parallelization, you must also specify **-qsmp** compiler option. In turn, you should specify the **-qsmp** option only in conjunction with one of these threadsafe invocation commands. When you specify **-qsmp**, the driver links in the libraries specified on the `smp libraries` line in the active stanza of the configuration file.

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

## POSIX Pthreads API support

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

## Specifying compiler options

Compiler options perform a variety of functions, such as setting compiler characteristics, describing the object code to be produced, controlling the diagnostic messages emitted, and performing some preprocessor functions.

You can specify compiler options in one or any combination of the following ways:

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

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

For more information about compiler options and their usage, see "Specifying options on the command line" in the *XL Fortran Compiler Reference*.

## Priority sequence of compiler options

It is possible for option conflicts and incompatibilities to occur when multiple compiler options are specified. To resolve these conflicts in a consistent fashion, the compiler usually applies the following general priority sequence to most options:

1. Directive statements in your source file *override* command-line settings
2. Command-line compiler option settings *override* configuration file settings
3. Configuration file settings *override* default settings

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

**Note:** Some compiler options, such as the **-I** option, do not follow the priority sequence described above. The compiler searches any directories specified with **-I** in the `xlf.cfg` file before it searches the directories specified with **-I** on the command-line. The **-I** option is cumulative rather than preemptive. Other options with cumulative behavior are **-R** and **-l** (lowercase L).

## XL Fortran input and output files

These file types are recognized by XL Fortran.

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

Table 12. Input file types

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

Table 13. Output file types

Filename extension	Description
a.out	Default name for executable file created by the compiler
.mod	Module symbol files
.smod <b>1</b>	Submodule symbol files
.lst	Listing files
.o	Object files
.s	Assembler files
.so	Shared object or library files
<b>Note:</b> <b>1</b> 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 the following command to compile file1.f and file3.f to produce the object files file1.o and file3.o. 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
```

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

## Dynamic and static linking

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

Dynamic linking means that the code for some external routines is located and loaded when the program is first run. When you compile a program that uses shared libraries, the shared libraries are dynamically linked to your program by default. Dynamically linked programs take up less disk space and less virtual memory if more than one program uses the routines in the shared libraries. During linking, they do not require any special precautions to avoid naming conflicts with library routines. They are designed to 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.

Because this form of linking is the default, you need no additional options to turn it on.

Static linking means that the code for all routines called by your program becomes part of the executable file.

Statically linked programs can be moved to 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. They also 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 together with any run time arguments on the command line.

### Canceling execution

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

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

## Setting runtime options

You can use environment variable settings to control certain runtime options and behaviors of applications created with the XL Fortran compiler. Some environment variables do not control actual runtime behavior, but they can have an impact on how your applications run.

For more information on environment variables and how they can affect your applications at run time, see the *XL Fortran Installation Guide*.

## Running compiled applications on other systems

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 PTF images, together with licensing and usage information, from the XL Fortran for Linux 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 **-qlinedebug** option to instruct the XL Fortran compiler to include debugging information in compiled output. For **-g**, you can also use different levels to balance between debug capability and compiler optimization. For more information about the debugging options, see "Error checking and debugging" in the *XL Fortran Compiler Reference*.

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

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

## Determining 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

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

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

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

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

Any references in this information to non-IBM 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:

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

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

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

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

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

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

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

#### COPYRIGHT LICENSE:

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

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1998, 2014.

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

---

## Trademarks and service marks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines 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 52
- .f and .F files 52
- .i files 52
- .lst files 52
- .mod files 52
- .o files 52
- .s files 52
- .S files 52
- .so files 52

## Numerics

- 64-bit environment 5

## A

- archive files 52
- assembler
  - source (.s) files 52
  - source (.S) files 52

## B

- backward 21
- Backward compatibility issues 21
- basic example, described x

## C

- code optimization 5
- compatibility 21
- compilation
  - sequence of activities 47
- compiler
  - controlling behavior of 51
  - invoking 48
  - running 48
- compiler directives
  - new or changed 30, 38
- compiler options
  - conflicts and incompatibilities 51
  - new or changed 16, 30, 38
  - specification methods 51
- compiling
  - SMP programs 50

## D

- debugger support 54
  - output listings 54
  - symbolic 7
- debugging 54
- debugging compiled applications 54
- debugging information, generating 54
- directives 40
- dynamic linking 53

## E

- editing source files 47
- executable files 52
- executing a program 53
- executing the linker 52

## F

- f2003 command
  - description 48
  - level of Fortran standard compliance 49
- f2008 command
  - description 48
  - level of Fortran standard compliance 49
- f77 command
  - description 48
  - level of Fortran standard compliance 49
- f90 command
  - description 48
  - level of Fortran standard compliance 49
- f95 command
  - description 48
  - level of Fortran standard compliance 49
- files
  - editing source 47
  - input 52
  - output 52
- fort77 command
  - description 48
  - level of Fortran standard compliance 49
- Fortran 2003
  - compiling programs written for 49
- Fortran 2008
  - compiling programs written for 49
- Fortran 90
  - compiling programs written for 49
- Fortran 95
  - compiling programs written for 49

## I

- input files 52
- invocation commands 48
- invoking a program 53
- invoking the compiler 48

## L

- language standards 3
- language support 3
- level of XL Fortran, determining 55
- libraries 52

- linking
  - dynamic 53
  - static 53
- linking process 52
- listings 52

## M

- migrate 21
- migration 21
  - source code 51
- mod files 52
- multiprocessor systems 6, 33, 41

## O

- object files 52
  - creating 52
  - linking 52
- OMP directives 33, 41
- OpenMP 6
- optimization
  - programs 5
- output files 52

## P

- parallelization 6, 33, 41
- performance
  - optimizing transformations 5
- POSIX Pthreads
  - API support 51
- problem determination 54
- programs
  - running 53

## R

- running the compiler 48
- runtime
  - libraries 52
- runtime environment 54
- runtime options 54

## S

- shared memory parallelization 6, 33, 41
- shared object files 52
- SMP
  - programs, compiling 50
- SMP programs 6
- source files 52
- source-level debugging support 7
- static linking 53
- symbolic debugger support 7

## T

- tools 4
  - cleanpdf utility 4
  - configuration file utility 4
  - mergepdf utility 4
  - new install configuration utility 4
  - new\_install utility 4
  - resetpdf utility 4
  - showpdf utility 4
  - xl\_f\_configure 4

## U

- utilities 4
  - cleanpdf 4
  - mergepdf 4
  - new\_install 4
  - resetpdf 4
  - showpdf 4
  - xl\_f\_configure 4

## X

- xl\_c.cfg file 51
- xl\_f command
  - description 48
- xl\_f\_r command
  - description 48
  - for compiling SMP programs 50
- xl\_f2003 command
  - description 48
  - level of Fortran standard compliance 49
- xl\_f2003\_r command
  - description 48
  - level of Fortran standard compliance 49
- xl\_f2008 command
  - description 48
  - level of Fortran standard compliance 49
- xl\_f2008\_r command
  - description 48
  - level of Fortran standard compliance 49
- xl\_f90 command
  - description 48
  - level of Fortran standard compliance 49
- xl\_f90\_r command
  - description 48
  - for compiling SMP programs 50
  - level of Fortran standard compliance 49
- xl\_f95 command
  - description 48
  - level of Fortran standard compliance 49
- xl\_f95\_r command
  - description 48
  - for compiling SMP programs 50
  - level of Fortran standard compliance 49





Product Number: 5765-J10; 5725-C75

Printed in USA

SC27-4252-00

