IBM XL Fortran Enterprise Edition for AIX, V11.1

Getting Started with XL Fortran

GC23-5835-00

IBM XL Fortran Enterprise Edition for AIX, V11.1

Getting Started with XL Fortran

GC23-5835-00

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 25.

First Edition

This edition applies to IBM XL Fortran Enterprise Edition for AIX, V11.1 (Program number 5724-S72) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Contents

About this document

This document contains overview and basic usage information for the IBM® XL Fortran Enterprise Edition for AIX®, V11.1 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 the XL Fortran compiler.

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 *IBM XL Fortran Enterprise Edition for AIX, V11.1 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 used in this document

Typographical conventions

The following table explains the typographical conventions used in this document.

Table 1. Typographical conventions

Typeface	Indicates	Example
bold	Lowercase commands, executable names, compiler options and directives.	If you specify **-O3**, the compiler assumes **-qhot=level=0**. To prevent all HOT optimizations with **-O3**, you must specify **-qnohot**.
italics	Parameters or variables whose actual names or values are to be supplied by the user. Italics are also used to introduce new terms.	The maximum length of the *trigger_constant* in fixed source form is 4 for directives that are continued on one or more lines.

Table 1. Typographical conventions (continued)

Typeface	Indicates	Example
`monospace`	Examples of program code, command strings, or user-defined names.	Also, specify the following runtime options before running the program, with a command similar to the following: `export XLFRTEOPTS="err_recovery=no: langlvl=90std"`

Syntax diagrams

Throughout this document, 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.

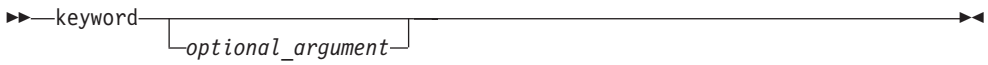
  The —►◄ symbol indicates the end of a command, directive, or statement.

  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):

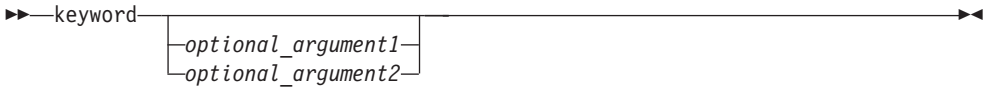
- Optional items are shown below the main path:



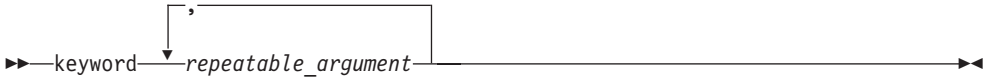
  Note: Optional items (not in syntax diagrams) are enclosed by square brackets ([and]). For example, `[UNIT=]u`
- If you can choose from two or more items, they are shown vertically, in a stack.

  If you *must* choose one of the items, one item of the stack is shown on the main path.

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



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

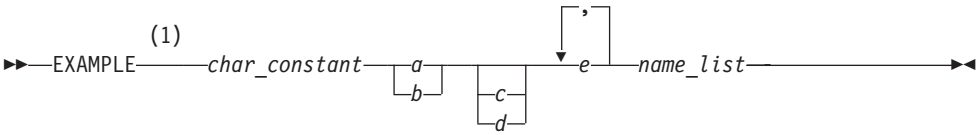


- The item that is the default is shown above the main path.

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

Sample syntax diagram

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



Notes:

1 IBM extension

Interpret the diagram as follows:

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

Examples

The examples in this document, 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.

Related information

The following sections provide information on documentation related to XL Fortran:

- "IBM XL Fortran publications"
- "Other IBM publications" on page ix

IBM XL Fortran publications

XL Fortran provides product documentation in the following formats:

- README files

  README files contain late-breaking information, including changes and corrections to the product documentation. 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 Enterprise Edition for AIX, V11.1 Installation Guide*.

- Information center

  The information center of searchable HTML files can be launched on a network and accessed remotely or locally. Instructions for installing and accessing the online information center are provided in the *IBM XL Fortran Enterprise Edition for AIX, V11.1 Installation Guide*. The information center is also viewable on the Web at http://publib.boulder.ibm.com/infocenter/comphelp/v9v111/index.jsp.

- PDF documents

  PDF documents are located by default in the /usr/lpp/xlf/doc/*LANG*/pdf/ directory, where *LANG* is one of en_US or ja_JP. The PDF files are also available on the Web at http://www.ibm.com/software/awdtools/fortran/xlfortran/library.

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

Table 2. XL Fortran PDF files

Document title	PDF file name	Description
IBM XL Fortran Enterprise Edition for AIX, V11.1 Installation Guide, GC23-5834-00	install.pdf	Contains information for installing XL Fortran and configuring your environment for basic compilation and program execution.
Getting Started with IBM XL Fortran Enterprise Edition for AIX, V11.1, GC23-5835-00	getstart.pdf	Contains an introduction to the XL Fortran product, with information on setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
IBM XL Fortran Enterprise Edition for AIX, V11.1 Compiler Reference, SC23-5833-00	cr.pdf	Contains information about the various compiler options and environment variables.

Table 2. XL Fortran PDF files (continued)

Document title	PDF file name	Description
IBM XL Fortran Enterprise Edition for AIX, V11.1 Language Reference, SC23-5832-00	lr.pdf	Contains information about the Fortran programming language as supported by IBM, including language extensions for portability and conformance to non-proprietary standards, compiler directives and intrinsic procedures.
IBM XL Fortran Enterprise Edition for AIX, V11.1 Optimization and Programming Guide, SC23-5836-00	opg.pdf	Contains information on advanced programming topics, such as application porting, interlanguage calls, floating-point operations, input/output, application optimization and parallelization, and the XL Fortran high-performance libraries.

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

More documentation related to XL Fortran including redbooks, white papers, tutorials, and other articles, is available on the Web at:

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

Other IBM publications

- *AIX Commands Reference, Volumes 1 - 6*, SC23-4888
- *AIX Technical Reference: Base Operating System and Extensions, Volumes 1 & 2*, SC23-4913
- *AIX National Language Support Guide and Reference*, SC23-4902
- *AIX General Programming Concepts: Writing and Debugging Programs*, SC23-4896
- *AIX Assembler Language Reference*, SC23-4923

  All AIX documentation is available at http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp.
- *ESSL for AIX V4.2 Guide and Reference*, SA22-7904, available at http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp

Technical support

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

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

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

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 document or any other XL Fortran documentation, send your comments by e-mail to compinfo@ca.ibm.com.

Be sure to include the name of the document, the part number of the document, 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 Enterprise Edition for AIX, V11.1 is an advanced, high-performance compiler that can be used for developing complex, computationally intensive programs, including interlanguage calls with C and C++ programs.

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

Commonality with other IBM compilers

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

IBM XL Fortran Enterprise Edition for AIX, V11.1 is part of a larger family of IBM C, C++, and Fortran compilers.

These compilers are derived from a common code base that shares compiler function and optimization technologies on a variety of platforms and programming languages, such as AIX, i5/OS®, selected Linux® distributions, z/OS®, and z/VM® operating systems. The common code base, along with compliance with international programming language standards, helps support consistent compiler performance and ease of program portability across multiple operating systems and hardware platforms.

Hardware and operating system support

V11.1 of XL Fortran supports AIX 5L™ for POWER™ V5.2 and V5.3. See the README file and "Before installing XL Fortran" in the *XL Fortran Installation Guide* for a complete list of requirements.

The compiler, its libraries, and its generated object programs will run on systems with the required software and disk space.

To take maximum advantage of the various supported hardware configurations, the compiler provides options to performance-tune applications specifically to the type of hardware that will be used to execute your compiled applications.

A highly configurable compiler

XL Fortran offers you a wealth of features to let you tailor the compiler to your own unique compilation requirements.

Compiler invocation commands

XL Fortran provides several different commands that you can use to invoke the compiler, for example, **xlf**, **xlf2003**, **xlf95**, and **xlf90**. Each invocation command is unique in that it instructs the compiler to tailor compilation output to meet a specific language level specification. Compiler invocation commands are provided to support all standardized Fortran language levels, and many popular language extensions as well.

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

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

Compiler options

You can choose from a large selection of compiler options to control compiler behavior. Different categories of options help you to debug your applications, optimize and tune application performance, select language levels and extensions for compatibility with non-standard features and behaviors supported by other Fortran compilers, and perform many other common tasks that would otherwise require changing the source code.

XL Fortran lets you specify compiler options through a combination of environment variables, compiler configuration files, command line options, and compiler directive statements embedded in your program source.

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

Custom compiler configuration files

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

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

See "Using custom compiler configuration files" on page 15 for more information.

Language standards compliance

The compiler supports the following programming language specifications for Fortran:

- ISO/IEC 1539-1:1991(E) and ANSI X3.198-1992 (referred to as Fortran 90 or F90)
- ISO/IEC 1539-1:1997 (referred to as Fortran 95 or F95)
- Extensions to the Fortran 95 standard:
  - Industry extensions that are found in Fortran products from various compiler vendors
  - Extensions specified in SAA® Fortran
- Most of the Fortran 2003 standard, except for derived type parameters, but including object-oriented programming
- Common Fortran language extensions defined by other compiler vendors, in addition to those defined by IBM

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

- OpenMP V2.5 extensions to support portable parallelized programming
- Language extensions to support vector programming

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

Source-code migration and conformance checking

XL Fortran helps protect your investment in your existing Fortran source code by providing compiler invocation commands that instruct the compiler to inspect your application for conformance to to a specific language level and warn you if it finds constructs and keywords that do not conform to the specified language level. You can also use the **-qlanglvl** compiler option to specify a given language level, and the compiler will issue warnings if language elements in your program source do not conform to that language level. Additionally, you can name your source files with common filename extensions such as .f77, .f90, f95, or .f03, then use the generic compiler invocations such as **xlf** or **xlf_r** to automatically select the appropriate language-level appropriate to the filename extension.

To protect your investments in existing source code, you can rebuild your FORTRAN 77, Fortran 90, Fortran 95, and Fortran 2003 source with XL Fortran and link them all into the same application. Similarly, object code or libraries compiled with previous versions of XL Fortran are still compatible with the newest XL Fortran compiler and runtime environment.

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

Tools and utilities

xlfndi This is a script you can use to perform an advanced installation of XL Fortran to a non-default directory location.

cleanpdf command
A command related to profile-directed feedback (PDF), **cleanpdf** removes all profiling information from the directory to which profile-directed feedback data is written.

mergepdf command
A command related to profile-directed feedback (PDF), **mergepdf** provides the ability to weight the importance of two or more PDF records when combining them into a single record. The PDF records must be derived from the same executable.

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

showpdf command
The **showpdf** command displays the call and block counts for all procedures executed in a profile-directed feedback training run (compilation under the options **-qpdf1** and **-qshowpdf**).

Program optimization

XL Fortran provides several compiler options that can help you control the optimization of your programs. With these options, you can:

- Select different levels of compiler optimizations
- Control optimizations for loops, floating point, and other types of operations

- Optimize a program for a particular class of machines or for a very specific machine configuration, depending on where the program will run

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

- Reduce the number of instructions executed for critical operations.
- Restructure generated object code to make optimal use of the PowerPC® architecture.
- Improve the usage of the memory subsystem.
- Exploit the ability of the architecture to handle large amounts of shared memory parallelization.

Significant performance improvements are possible with relatively little development effort because the compiler is capable of sophisticated program analysis and transformation. Moreover, XL Fortran enables programming models, such as OpenMP, which allow you to write high-performance parallel code.

For more information, see:

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

64-bit object capability

The XL Fortran compiler's 64-bit object capability addresses increasing demand for larger storage requirements and greater processing power. The AIX operating system provides an environment that allows you to develop and execute programs that exploit 64-bit processors through the use of 64-bit address spaces.

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

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

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

XL Fortran supports 64-bit mode mainly through the use of the **-q64** and **-qarch** compiler options. This combination determines the bit mode and instruction set for the target architecture.

For more information, see "Using XL Fortran in a 64-Bit Environment" in the *XL Fortran Compiler Reference*.

Shared memory parallelization

XL Fortran supports application development for multiprocessor system architectures. You can use any of the following methods to develop your parallelized applications with XL Fortran:

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

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

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

OpenMP directives

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

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

OpenMP directives address three important issues of parallelizing an application:

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

XL Fortran supports the OpenMP API Version 2.5 specification.

For more information, see:

- "Optimizing XL compiler applications" in the *XL Fortran Optimization and Programming Guide*
- www.openmp.org

Diagnostic listings

The compiler output listing can provide important information to help you more efficiently develop and debug your application. Listing information is organized into optional sections that you can include or omit. For more information about the applicable compiler options and the listing itself, refer to ″Understanding XL Fortran compiler listings″ in the *XL Fortran Compiler Reference*.

Symbolic debugger support

You can instruct XL Fortran to include debugging information in your compiled objects. That information can be examined by **dbx** or any other symbolic debugger that supports the AIX XCOFF executable format to help you debug your programs.

Chapter 2. What's new for IBM XL Fortran Enterprise Edition for AIX, V11.1

This section describes new features and enhancements in IBM XL Fortran Enterprise Edition for AIX, V11.1.

Enhanced support for Fortran 2003

Beginning with the V8.1 release, XL Fortran has delivered many Fortran 2003 standard features, including:

- BIND(C) for portable interoperability with C code
- Allocatable objects beyond just Fortran 95 arrays
- Stream I/O support
- ASSOCIATE and ENUM statements
- READ with BLANK= and PAD= specifiers
- WRITE with DELIM= specifier
- IEEE and Fortran environment modules

XL Fortran V11.1 builds on that work to arrive at the most complete Fortran 2003 implementation currently available, with Derived Type Parameters being the only major feature not yet implemented.

New Fortran 2003 compiler invocations and file types

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

- xlf2003
- xlf2003_r (for threaded applications)
- f2003

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

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

In addition to new compiler invocations, this release of XL Fortran also adds support for new filename extensions:

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

New compiler option -qxlf2003

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

When compiling with the **xlf2003**, **xlf2003_r**, or **f2003** 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 **-qxlf2003** compiler option also includes several other suboptions to provide backward compatibility with earlier versions of XL Fortran. See **-qxlf2003** in the *XL Fortran Compiler Reference* for more information.

Additional Fortran 2003 enhancements

Significant enhancements introduced with XL Fortran V11.1 include:

- Implementation of the full Fortran 2003 object-oriented programming model, including:
  - type extension
  - type-bound procedures
  - type finalization
  - polymorphism and runtime type determination including the SELECT TYPE construct
  - abstract and generic interfaces
  - declaration of abstract types and deferred bindings
  - PASS attribute
- I/O enhancements
  - User-defined derived type I/O
  - New I/O specifiers including SIGN= and DECIMAL= (DC and DP edit descriptors)
  - Asynchonous I/O as defined by Fortran 2003 including the WAIT statement
  - User-specifed control of rounding during format conversion using the ROUND= specifier and new edit descriptors
  - Handling of IEEE infinity and not-a-number in REAL and COMPLEX editing
  - Support of PAD= specifier on INQUIRE operations
- Scoping and data manipulation enhancements
  - Renaming of defined operators on USE statements
  - Fortran 2003 VOLATILE statement
  - MAX, MIN, MAXLOC, MINLOC, MAXVAL, and MINVAL instrinsics for character types
  - COMPLEX literals
  - Pointer assignment and initialization expression enhancements
  - Improved structure constructors
  - Allocatable enhancements including resizing on assignment and MOVE_ALLOC intrinsic
  - Explicit type specification in an array constructor
- Subroutine enhancements
  - Generic interface and operator assignment
  - VALUE attribute for characters of length greater than one and derived types with allocatable components
  - Procedure pointers, procedure declaration statement, and procedure pointers as derived type components

  - Generalization of the MODULE PROCEDURE statement
  - Deferred CHARACTER length
- Intrinsic Function Enhancements
  - Allow REAL type for COUNT_RATE argument of SYSTEM_CLOCK
  - Allow boz-literal constants on INT, REAL, CMPLX, and DBL intrinsics
  - Allow a new KIND argument on all instrinsics mandated by Fortran 2003
  - Returning signed zero results from the ATAN2, LOG, and SQRT intrinsics
  - Added SELECTED_CHAR_KIND intrinsic
- Other enhancements
  - Enhanced STOP statement
  - Increased the maximum number of continuation lines

Architecture and processor support

The **-qarch** and **-qtune** compiler options control the code generated by the compiler. These compiler options adjust the instructions, scheduling, and other optimizations to give the best performance for a specified target processor or range of processors.

New default settings for -qarch, -qtune

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

- -qarch=ppc
- -qtune=balanced

The **-qtune=balanced** suboption is new for this release, and becomes the default **-qtune** setting when certain **-qarch** settings are specified. Using **-qtune=balanced** instructs the compiler to tune generated code for optimal performance across a range of recent processor architectures, including POWER6™.

Important: The change to the **-qarch** default suboption setting can affect the results of floating point arithmetic computations with REAL(4) data types in your programs.

The **-qarch=com** default used in the previous release of the compiler caused such computations to be performed using double precision instructions followed by rounding. The new **-qarch=ppc** default instructs the compiler to generate code that uses short floating point instructions. The difference in computational method can affect the precision of computational results.

To achieve the behavior of the previous **-qarch=com** default, specify the new **-qfloat=nosingle** compiler option when compiling your application.

New support for POWER6 processors

XL Fortran 11.1 expands the list of **-qarch** and **-qtune** suboptions to support the newly-available POWER6 processors.

The following **-qarch** and **-qtune** options are now available:

- -qarch=pwr6
- -qarch=pwr6e

- -qtune=pwr6

The **-qipa** compiler option also adds a new architecture cloning suboption to support interprocedural analysis (IPA) optimizations on POWER6 processors:

- -qipa=clonearch=pwr6

Support removed for selected processors

XL Fortran 11.1 removes support for processor architectures not supported by AIX V5.2, such as POWER, POWER2™, and PowerPC 601. As a result, the following **-qarch** and **-qtune** suboption settings are no longer supported.

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

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

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

Performance and optimization

Many new features and enhancements fall into the category of performance and optimization tuning.

Performance-related compiler options and directives

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

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

Table 3. Performance-related compiler options and directives

Option/directive	Description
-qfloat= fenv | nofenv	These new **-qfloat** suboptions inform the compiler if code has a dependency on the floating-point hardware environment, such as explicitly reading or writing the floating-point status and control register. Specifying **-qfloat=nofenv** indicates that there is no dependency on the hardware environment, allowing the compiler to perform aggressive optimizations.
-qfloat= hscmplx | nohscmplx	Specifying **-qfloat=hscmplx** improves optimization of operations involving complex division and complex absolute values.
-qfloat= rngchk | norngchk	Specifying **-qfloat=rngchk** enables range checking on input arguments for software divide and inlined sqrt operations. Specifying **-qfloat=norngchk** instructs the compiler to skip range checking, allowing for better performance in certain circumstances. Specifying the **-qnostrict** compiler option sets **-qfloat=norngchk**.

Table 3. Performance-related compiler options and directives (continued)

Option/directive	Description
-qfloat= single | nosingle	Specifying **-qfloat=single** instructs the compiler to compute single-precision floating-point values using single-precision arithmetic instructions supported by all current PowerPC processors. Use **-qfloat=nosingle** if you need to preserve the computational behavior in applications originally compiled for earlier processors, such as POWER and POWER2 processors. You may also need to specify **-qfloat=norndsngl** to obtain the same computational results.
-qipa=clonearch=pwr6	The **-qipa=clonearch** compiler option now includes a new **pwr6** suboption to support interprocedural analysis (IPA) optimizations on POWER6 processors.
-qipa=threads= **[auto | noauto |** *number***]**	This new **-qipa** suboption lets you specify how many threads the compiler will assign to code generation during the second IPA pass.
-qminimaltoc | -qnominimaltoc	Specifying **-qminimaltoc** helps avoid toc overflow conditions in 64-bit compilations by placing toc entries into a separate data section for each object file.
-qpdf	The **-qpdf** option can now be used to provide profile-directed feedback on specific objects. See "Object level profile-directed feedback" in the *XL Fortran Optimization and Programming Guide* for more information.
-qsmp= threshold=*n*	When **-qsmp=auto** is in effect, this new suboption lets you specify the amount of work required in a loop before the compiler will consider it for automatic parallelization.
!IBM EXPECTED_VALUE(*param***,** *value***)**	Use the **!IBM EXPECTED_VALUE** directive to specify a value that a parameter passed in a function call is most likely to take at run time. The compiler can use this information to perform certain optimizations, such as function cloning and inlining.

Directives new for this release

This section lists directives that are new for this release. For more information on directives provided by XL Fortran, see "Hardware-specific directives" in the *XL Fortran Language Reference*.

PowerPC cache control

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

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

POWER6 prefetch extensions and cache control

The POWER6 processor has cache control and stream prefetch extensions with support for store stream prefetch and prefetch depth control. XL Fortran provides the following new directives to provide direct programmer access to these instructions.

- DCBFL(*variable*); ! pwr6 - Data Cache Block Flush from L1 data cache only
- PROTECTED_UNLIMITED_STREAM_SET_FORWARD(*prefetch_variable*, *stream_id*) ! pwr5 and pwr6
- PROTECTED_UNLIMITED_STREAM_SET_BACKWARD(*prefetch_variable*, *stream_id*) ! pwr5 and pwr6
- PROTECTED_UNLIMITED_STORE_STREAM_SET_FORWARD(*prefetch_variable*, *stream_id*) ! pwr6
- PROTECTED_UNLIMITED_STORE_STREAM_SET_BACKWARD(*prefetch_variable*, *stream_id*) ! pwr6
- PROTECTED_STORE_STREAM_SET_FORWARD(*prefetch_variable*, *stream_id*) ! pwr6
- PROTECTED_STORE_STREAM_SET_BACKWARD(*prefetch_variable*, *stream_id*) ! pwr6
- PROTECTED_STREAM_COUNT_DEPTH(*unit_count*, *prefetch_depth*, *stream_id*) ! pwr6

Other new or changed compiler options

Compiler options can be specified on the command line or through directives embedded in your application source files. See the *XL Fortran Compiler Reference* for detailed descriptions and usage information for these and other compiler options.

Table 4. Other new or changed compiler options

Option/directive	Description
-qalias_size=*bytes*	The **-qalias_size** option helps you avoid memory fragmentation in aliasing tables by letting you specify an appropriate initial size for those tables. This option can also be specified in your program source with **@PROCESS ALIAS_SIZE(***bytes***)**.
-qdescriptor= **v1**|**v2**	The **-qdescriptor** option lets you select the XL Fortran internal descriptor data structure format used in your compiled applications. The default **v1** suboption uses a data structure format that provides backwards compatibility with objects compiled with XL Fortran V10.1 and earlier. Regardless of what **-qdescriptor** setting is in effect, applications containing object-oriented constructs will use the **v2** suboption data structure format for those constructs, and will not be compatible with objects compiled with XL Fortran V10.1 and earlier. You should consider explicitly specifying the **v2** suboption if you are compiling new applications that will not need to interact with objects compiled with XL Fortran V10.1 and earlier.
-qoptdebug|**-qnooptdebug**	When used with optimization levels of **-O3** or higher, the new **-qoptdebug** option instructs the compiler to produce optimized pseudocode that can be read by a symbolic debugger.
-qreport	When used together with compiler options that enable automatic parallelization or vectorization, the **-qreport** option now produces a pseudo-code listing showing how program loops are parallelized and vectorized. The report also provides diagnostic information if the compiler is not able to parallelize or vectorize a given loop.

Table 4. Other new or changed compiler options (continued)

Option/directive	Description
-qsaveopt | **-qnosaveopt**	In previous releases, the **-qsaveopt** option stored the command line options used to compile a file into the resulting object file. In this release, the information stored in the object file is expanded to also include version and level information for each compiler component invoked during compilation.
-qsmp=stackcheck	This new **-qsmp** suboption instructs the compiler to check for stack overflow by slave threads at run time, and issue a warning if the remaining stack size is less than the number of bytes specified by the **stackckeck** option of the XLSMPOPTS environment variable.
-qversion=verbose	The **-qversion** option adds a new **verbose** suboption. Specifying **-qversion=verbose** instructs the compiler to display the version and level information for each compiler component invoked during compilation.

Chapter 3. Setting up and customizing XL Fortran

For complete prerequisite and installation information, please refer to *XL Fortran Installation Guide*.

Using custom compiler configuration files

A default compiler configuration file is created during XL Fortran compiler installation, and you can directly modify this configuration file to add default options for specific needs. However, if you later apply updates to the compiler, you will also need to reapply all of your modifications to the newly installed configuration file.

You can avoid this by creating your own custom compiler configuration files. The compiler now has the ability to recognize and resolve compiler settings you specify in your custom configuration files together with compiler settings specified in the default configuration file.

If you instruct the compiler to use a custom configuration file, the compiler will examine and process the settings in that custom configuration file before looking at settings in the default system configuration file. Compiler updates that may later affect settings in the default configuration file will not affect the settings in your custom configuration files.

See "Using custom compiler configuration files" in the *XL Fortran Compiler Reference* for more information.

Determining what level of XL Fortran is installed

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

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

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

```
xlf -qversion=verbose
```

Chapter 4. Developing applications with XL Fortran

Basic Fortran application development consists of repeating cycles of editing, compiling and linking (by default a single step combined with compiling), and running.

Note:
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 the following activities in sequence. For link time optimizations, some activities will be executed more than once during a compilation. As each program runs, the results are sent to the next step in the sequence.

1. Preprocessing of source files
2. Compilation, which may consist of the following phases, depending on what compiler options are specified:
   a. Front-end parsing and semantic analysis
   b. Loop transformations
   c. High-level optimization
   d. Low-level optimization
   e. Register allocation
   f. Final assembly
3. Program assembly for **.s** files and for **.S** files after they are preprocessed
4. Object linking to create an executable application

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

Editing Fortran source files

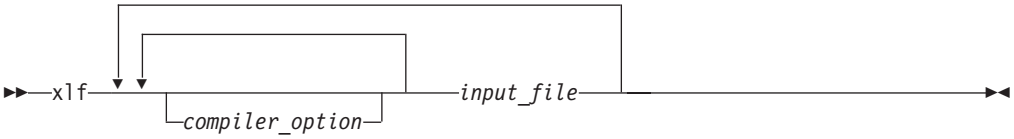
To create Fortran source programs, you can use any text editor available to your system, such as **vi** or **emacs**. Source programs must be saved using a recognized file name suffix. See the "XL Fortran input and output files" on page 21 for a list of suffixes recognized by XL Fortran.

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

Invoking the compiler

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



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.

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

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

Compiling Fortran 77 programs

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

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

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

Compiling Fortran 95, or Fortran 90 programs

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

f95, xlf95 Fortran 95

f90, xlf90 Fortran 90

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

I/O formats are slightly different between these commands and the other commands. I/O formats for the **xlf95** invocation are also different from those of **xlf90**. We recommend that you switch to the Fortran 95 formats for data files whenever possible.

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

For full Fortran 90 compliance:

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

For full Fortran 95 compliance:

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

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

For full Fortran 90 compliance:

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

For full Fortran 95 compliance:

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

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

Compiling Fortran 2003 programs

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

f2003, xlf2003 Fortran 2003

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

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

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

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

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

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

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

-qxlf2003 compiler option

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

When compiling with the **xlf2003**, **xlf2003_r**, or **f2003** compiler invocations, the default setting is **-qxlf2003=polymorphic**. This setting instructs the compiler to allow polymorphic items such as the CLASS type specifier and SELECT TYPE construct in your Fortran application source.

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

Compiling parallelized XL Fortran applications

XL Fortran provides thread safe compiler invocation commands that you can use when compiling parallelized applications for use in multiprocessor environments. These invocations are similar to their corresponding base compiler invocations, except that they link and bind compiled objects to thread safe components and libraries.

The generic XL Fortran thread safe compiler invocation is:

- xlf_r

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

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

POSIX Pthreads API support

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

You can use invocation commands (which use corresponding stanzas in the xlf.cfg configuration file) to compile and then link your programs with either the 1003.1-1996 standard or the Draft 7 interface libraries. For more information on thread safe **_r** and **_r7** compiler invocation variants, refer to "Invoking the Compiler" in the *XL Fortran Compiler Reference*.

Specifying compiler options

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

You can specify compiler options:

- On the command-line with command-line compiler options
- In your source code using directive statements
- In a makefile
- In the stanzas found in a compiler configuration file
- Or by using any combination of these techniques

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

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

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

Note: Some compiler options do not follow the priority sequence described above.

For example, the **-I** compiler option is a special case. The compiler searches any directories specified with **-I** in the xlf.cfg file before it searches the directories specified with **-I** on the command-line. The option is cumulative rather than preemptive.

See the *XL Fortran Compiler Reference* for more information about compiler options and their usage.

You can also pass compiler options to the linker, assembler, and preprocessor. See "Specifying options on the command line" in the *XL Fortran Compiler Reference* for more information about compiler options and how to specify them.

XL Fortran input and output files

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

Table 5. Input file types

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

Table 6. Output file types

Filename extension	Description
a.out	Default name for executable file created by the compiler

Table 6. Output file types (continued)

Filename extension	Description
.mod	Module symbol files
.lst	Listing files
.o	Object files
.s	Assembler files
.so	Shared object files

Linking your compiled applications with XL Fortran

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

```
xlf95 file1.f file2.o file3.f
```

compiles and produces the object files `file1.o` and `file3.o`, then all object files (including `file2.o`) are submitted to the linker to produce one executable.

Compiling and linking in separate steps

To produce object files that can be linked later, use the **-c** option.

```
xlf95 -c file1.f               # Produce one object file (file1.o)
xlf95 -c file2.f file3.f       # Or multiple object files (file2.o, file3.o)
xlf95 file1.o file2.o file3.o  # Link object files with default libraries
```

Linking new objects with existing ones

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

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

Relinking an existing executable file

On AIX, the linker will accept executable files as input, so you can link an existing executable file with updated object files. You cannot, however, relink executable files that were previously linked using the **-qipa** option.

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

```
xlf95 -omansion front_door.f entry_hall.f parlor.f sitting_room.f \
  master_bath.f kitchen.f dining_room.f pantry.f utility_room.f

vi kitchen.f # Fix problem in OVEN subroutine

xlf95 -o newmansion kitchen.f mansion
```

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

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

Dynamic and static linking

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

- Dynamic linking means that the code for some external routines is located and loaded when the program is first run. When you compile a program that uses shared libraries, the shared libraries are dynamically linked to your program by default.

  Dynamically linked programs take up less disk space and less virtual memory if more than one program uses the routines in the shared libraries. During linking, they do not require any special precautions to avoid naming conflicts with library routines. They may perform better than statically linked programs if several programs use the same shared routines at the same time. They also allow you to upgrade the routines in the shared libraries without relinking.

  Because this form of linking is the default, you need no additional options to turn it on.
- Static linking means that the code for all routines called by your program becomes part of the executable file.

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

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

Running your compiled application

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

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

You should avoid giving your program executable file the same name as system or shell commands, such as **test** or **cp**, as you could accidentally execute the wrong command. If you do decide to name your program executable file with the same name as a system or shell command, you should execute your program by specifying the path name to the directory in which your program executable file resides, such as `./test`.

Canceling execution

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

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

Setting run time options

You can use environment variable settings to control certain run time options and behaviors of applications created with the XL Fortran compiler. Other environment variables do not control actual run time behavior, but 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 *XL Fortran Installation Guide*.

Running compiled applications on other systems

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

If you want to run an application developed with the XL Fortran compiler on another system that does not have the compiler installed, you will need to install a runtime environment on that system.

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

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

XL Fortran compiler diagnostic aids

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

The XL Fortran runtime will also issue messages for certain unsupported operations, particularly I/O-related, when you run your application.

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

Specifying the **-g** or **-qlinedebug** compiler options at compile time instructs the XL Fortran compiler to include debugging information in compiled output.

You can then use **dbx** or any other symbolic debugger that supports the AIX XCOFF executable format to step through and inspect the behavior of your compiled application.

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

Notices

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

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

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

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1998, 2007. All rights reserved.

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

Trademarks and service marks

Company, product, or service names identified in the text may be trademarks or service marks of IBM or other companies. Information on the trademarks of International Business Machines Corporation in the United States, other countries, or both is located at http://www.ibm.com/legal/copytrade.shtml.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

Other company, product, and service names may be trademarks or service marks of others.

Index

Special characters

Numerics

A

C

D

E

F

I

L

M

O

P

R

S

T

U

V

X

Program Number: 5724-S72

GC23-5835-00