

IBM XL Fortran Enterprise Edition V10.1 for AIX



Compiler Reference

IBM XL Fortran Enterprise Edition V10.1 for AIX



Compiler Reference

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 313.

First Edition (October 2005)

This edition applies to IBM® XL Fortran Enterprise Edition V10.1 for AIX® (Program 5724-M13) 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.

IBM welcomes your comments. You can send your comments electronically to the network ID listed below. Be sure to include your entire network address if you wish a reply.

- Internet: compinfo@ca.ibm.com

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1990, 2005. All rights reserved.

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

Contents

About this document	vii
Who should read this document	vii
How to use this document	vii
How this document is organized	vii
Conventions and terminology used in this document.	viii
Typographical conventions	viii
How to read syntax diagrams	viii
How to read syntax statements	x
Examples	xi
Notes on the terminology used	xi
Related information.	xi
IBM XL Fortran documentation	xi
Additional documentation	xii
Related documentation	xiii
Standards documents	xiii
Technical support	xiii
How to send your comments	xiv
Chapter 1. Introduction	1
Chapter 2. Overview of XL Fortran features.	3
Hardware and operating-system support	3
Language support	3
Migration support	4
Source-code conformance checking	4
Highly configurable compiler.	4
Diagnostic listings	5
Symbolic debugger support	5
Program optimization	5
Chapter 3. Setting up and customizing XL Fortran.	7
Where to find installation instructions	7
Using the compiler on a network file System	7
Correct settings for environment variables	8
Environment variable basics	8
Environment variables for national language support	9
LIBPATH:Setting library search paths.	10
PDFDIR: Specifying the directory for PDF profile information	10
TMPDIR: Specifying a directory for temporary files	10
XLFSRATCH_unit: Specifying names for scratch files	11
XLFUNIT_unit: Specifying names for implicitly connected files	11
Customizing the configuration file.	11
Attributes	12
Determining which level of XL Fortran is installed	14
Upgrading to XL Fortran Version 10	15
Things to note in XL Fortran Version 10	15

Avoiding or fixing upgrade problems.	16
Running two levels of XL Fortran	18

Chapter 4. Editing, compiling, linking, and running XL Fortran programs **19**

Editing XL Fortran source files	19
Compiling XL Fortran programs	20
Compiling XL Fortran Version 2 programs	21
Compiling Fortran 90 or Fortran 95 programs	21
Compiling XL Fortran SMP programs	22
Compilation order for Fortran programs.	23
Canceling a compilation	23
XL Fortran input files	23
XL Fortran Output files	25
Scope and precedence of option settings.	26
Specifying options on the command line.	26
Specifying options in the source file	27
Passing command-line options to the "ld" or "as" commands.	28
Compiling for specific architectures	28
Passing Fortran files through the C preprocessor	29
cpp Directives for XL Fortran programs	30
Passing options to the C preprocessor	30
Avoiding preprocessing problems	30
Linking XL Fortran programs	31
Compiling and linking in separate Steps.	31
Linking 32-bit SMP object files using the ld command	31
Linking 64-bit SMP object files using the ld command	32
Linking 32-bit non-SMP object files using the ld command	33
Linking 64-bit non-SMP object files using the ld command	34
Passing options to the ld command	34
Checking for interface errors at link time	34
Linking new objects with existing ones	35
Relinking an existing executable file	35
Dynamic and Static Linking.	36
Avoiding naming conflicts during linking	36
Running XL Fortran programs	37
Canceling execution	38
Running previously compiled programs.	38
Compiling and executing on different systems.	38
POSIX Pthreads binary compatibility	38
Run-time libraries for POSIX pthreads support	39
Selecting the language for run-time messages	40
Setting run-time options	40
Other environment variables that affect run-time behavior	49
XL Fortran run-time exceptions.	49

Chapter 5. XL Fortran compiler-option reference. **51**

Summary of the XL Fortran compiler options	51
--	----

Options that control input to the compiler . . .	52	-qenum option	142
Options that specify the locations of output files	54	-qescape option.	143
Options for performance optimization	54	-qessl Option	144
Options for error checking and debugging . . .	59	-qextchk option.	145
Options that control listings and messages . . .	62	-qextern option.	146
Options for compatibility.	64	-qextname option	147
Options for floating-point processing	71	-qfdpr option	149
Options that control linking	72	-qfixed option	150
Options that Control Other Compiler Operations	73	-qflag option	151
Options that are obsolete or not recommended	74	-qfloat option	152
Detailed descriptions of the XL Fortran compiler		-qflttrap option.	154
options	76	-qfree option	156
# option	77	-qfullpath option	157
-l option	78	-qhalt option	158
-B option	79	-qhot option.	159
-b64 option	80	-qieee option	162
-bdynamic, -bshared, and -bstatic Options	81	-qinit option.	163
-bhalt option	83	-qinitauto option	164
-bloadmap option	84	-qintlog option	166
-bmaxdata, -bmaxstack Options.	85	-qintsize option.	167
-brtl option	86	-qipa option	169
-bshared option	87	-qkeepparm option	176
-bstatic option	88	-qlanglvl option	177
-C option	89	-qlargepage option	179
-c option	90	-qlibansi option.	180
-D option	91	-qlibessl option.	181
-d option	92	-qlibposix option	182
-F option	93	-qlinedebug option	183
-g option	94	-qlist option	184
-I option	95	-qlistopt option.	185
-k option	96	-qlog4 option	186
-L option	97	-qmaxmem option.	187
-l option	98	-qmbcs option	189
-NS option.	99	-qmixed option.	190
-O option.	100	-qmoddir option	191
-o option	102	-qmodule option	192
-p option	103	-qnoprint option	193
-Q option.	104	-qnullterm option	194
-q32 option	105	-qobject option	195
-q64 option	106	-qonetrip option	196
-qalias option	109	-qoptimize option	197
-qalign option	112	-qpdf option.	198
-qarch option	114	-qphsinfo option	202
-qassert option	119	-qpig option	204
-qattr option.	120	-qport option	205
-qautodbl option	121	-qposition option	207
-qcache option	124	-qprefetch option	208
-qcclines option.	126	-qqcount option	209
-qcheck option	127	-qrealsize option	210
-qci option	128	-qrecur option	212
-qcompact option	129	-qreport option	213
-qcr option	130	-qsaa option	215
-qctyplss option	131	-qsave option	216
-qdbg option	133	-qsaveopt option	217
-qddim option	134	-qselk option	218
-qdirective option	135	-qshowpdf option	219
-qdirectstorage option	137	-qsigtrap option	220
-qdlines option	138	-qsmallstack option	221
-qdpc option	139	-qsmp option	222
-qdpcl option	140	-qsource option.	227
-qenablevmx option	141	-qspillsize option	228

-qstacktemp option	229
-qstrict option	230
-qstrictieemod option	231
-qstrict_induction option	232
-qsuffix option	233
-qsuppress option	234
-qswapomp option	236
-qtbtable option	238
-qthreaded option	239
-qtune option	240
-qundef option	243
-qunroll option	244
-qunwind option	245
-qvecnvool option	246
-qversion option	247
-qwarn64 option	248
-qxflag=dvz option	249
-qxflag=oldtab option.	250
-qxflag=xalias option	251
-qxlf77 option	252
-qxlf90 option	254
-qxlines option	256
-qxref option	258
-qzerosize option	259
-S option	260
-t option	261
-U option.	262
-u option	263
-v option	264
-V option.	265
-W option	266
-w option.	268
-y option	269

Chapter 6. Using XL Fortran in a 64-Bit Environment 271

64-Bit Large Data Type support	271
64-Bit thread support.	272
Compiler options for the 64-Bit environment	272
Default bit mode	272
Module support	273

Chapter 7. Problem determination and debugging. 275

Understanding XL Fortran error messages.	275
Error severity	275
Compiler return code.	276
Run-time return code.	276
Understanding XL Fortran messages	276
Limiting the number of compile-time messages	277
Selecting the language for messages	277
Fixing installation or system environment problems	279
Fixing compile-time problems	279
Duplicating extensions from other systems	280
Isolating problems with individual compilation units	280
Compiling with thread-safe commands.	280
Running out of machine resources	280
Fixing link-time problems	280
Fixing run-time problems	281

Duplicating extensions from other systems	281
Mismatched sizes or types for arguments	281
Working around problems when optimizing	281
Input/Output errors	282
Tracebacks and core dumps	282
Debugging a Fortran 90 or Fortran 95 program	282
A Sample dbx session for an XL Fortran program	282
Problem with dynamic memory allocation.	283
Using debug memory routines for XL Fortran	287
The libhm.a library	287
The libhmd.a library	289
Environment variables	291

Chapter 8. Understanding XL Fortran compiler listings 295

Header section	295
Options section.	295
Source section	296
Error messages	296
Transformation report section	297
Attribute and cross-reference section	298
Object section	299
File table section	299
Compilation unit epilogue Section	299
Compilation epilogue Section	299

Chapter 9. Fortran-Related AIX commands 301

Working with object-code archives (ar)	301
Printing output files with Fortran ASA carriage controls (asa)	301
Splitting subprograms into individual files (fsplit)	301
Automating large, complex compilations (make)	302
Run-time profiling (prof, gprof)	302
Translating programs into RATFOR (struct)	302
Displaying information inside binary files (what)	302

Appendix A. XL Fortran technical information 303

The compiler phases	303
External Names in theXL FortranShared Libraries	303
The XL Fortran run-time environment	303
External names in the run-time environment	304
Technical details of the -qfloat=hsflt option	304
Implementation details for -qautodbl promotion and padding	305
Terminology.	305
Examples of storage relationships for -qautodbl suboptions	306

Appendix B. XL Fortran internal limits 311

Notices 313

Programming interface information	315
Trademarks and service marks	315

Glossary 317

Index 327

About this document

This document describes the IBM® XL Fortran Enterprise Edition V10.1 for AIX® compiler and explains how to set up the compilation environment and how to compile, link, and run programs written in the in the Fortran language. This guide also contains cross-references to relevant topics of other reference guides in the XL Fortran documentation suite.

Who should read this document

This document is for anyone who wants to work with the XL Fortran compiler, is familiar with the AIX operating system, and who has some previous Fortran programming experience. Users new to XL Fortran can use this document to find information on the capabilities and features unique to XL Fortran. This document can help you understand what the features of the compiler are, especially the options, and how to use them for effective software development.

How to use this document

While this document covers information about configuring the compiler, and compiling, linking and running XL Fortran programs, it does not include information on the following topics, which are covered in other documents:

- Installation, system requirements, last-minute updates: see the *XL Fortran Installation Guide* and product README.
- Overview of XL Fortran features: see the *Getting Started with XL Fortran Enterprise Edition V10.1 for AIX* .
- Syntax, semantics, and implementation of the XL Fortran programming language: see the *XL Fortran Language Reference*.
- Optimizing, porting, OpenMP/ SMP programming: see the *XL Fortran Optimization and Programming Guide*.
- Operating system commands related to the use of the compiler: *AIX Commands Reference* and the AIX information center.

How this document is organized

This document starts with an overview of the compiler and provides information on the tasks you need to do before invoking the compiler. It then continues with reference information about the compiler options and debugging problems.

This reference includes the following topics:

- Chapter 1, “Introduction” through Chapter 4, “Editing, compiling, linking, and running XL Fortran programs” discuss setting up the compilation environment and the environment variables that you need for different compilation modes, customizing the configuration file, the types of input and output files, compiler listings and messages and information specific to invoking the preprocessor and linkage editor.
- Chapter 5, “XL Fortran compiler-option reference” contains a table summarizing the compiler options by their functional category. You can search for options by their name, or alternatively use the links in the functional category tables and look up options according to their functionality. This section also includes

individual descriptions of the compiler options sorted alphabetically. Descriptions provide examples and list related topics.

- Chapter 6, “Using XL Fortran in a 64-Bit Environment” discusses application development for the 64-bit environment.
- Chapter 7, “Problem determination and debugging” through Chapter 9, “Fortran-Related AIX commands” address debugging, understanding compiler listings, and Fortran-related AIX commands.
- Appendix A, “XL Fortran technical information” and Appendix B, “XL Fortran internal limits” provide information that advanced programmers may need to diagnose unusual problems and run the compiler in a specialized environment.

Conventions and terminology 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	Commands, executable names, and compiler options.	By default, if you use the -qsmp compiler option in conjunction with one of these invocation commands, the option -qdirective=IBM*:SMP\$:SOMP:IBMP:IBMT will be on.
<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.	The maximum length of the <i>trigger_constant</i> in fixed source form is 4 for directives that are continued on one or more lines.
UPPERCASE bold	Fortran programming keywords, statements, directives, and intrinsic procedures.	The ASSERT directive applies only to the DO loop immediately following the directive, and not to any nested DO loops.
lowercase bold	Lowercase programming keywords and library functions, compiler intrinsic procedures, file and directory names, examples of program code, command strings, or user-defined names.	If you call omp_destroy_lock with an uninitialized lock variable, the result of the call is undefined.

How to read syntax diagrams

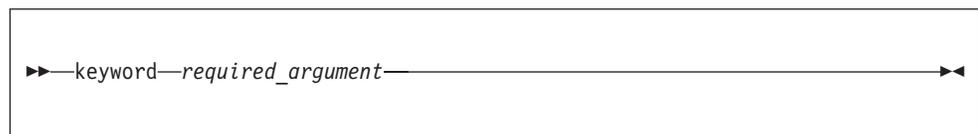
Throughout this document, diagrams illustrate XL Fortran syntax. This section will help you to interpret and use those diagrams.

If a variable or user-specified name ends in *_list*, you can provide a list of these terms separated by commas.

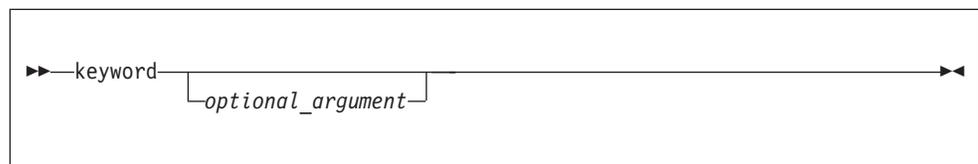
You must enter punctuation marks, parentheses, arithmetic operators, and other special characters as part of the syntax.

- Read syntax diagrams from left to right and from top to bottom, following the path of the line:

- The \blacktriangleright symbol indicates the beginning of a statement.
 - The \longrightarrow symbol indicates that the statement syntax continues on the next line.
 - The \blacktriangleright symbol indicates that a statement continues from the previous line.
 - The $\longrightarrow\blacktriangleleft$ symbol indicates the end of a statement.
 - 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.
 - IBM XL Fortran extensions to and implementations of language standards are marked by a number in the syntax diagram with an explanatory note immediately following the diagram.
- Required items are shown on the horizontal line (the main path):

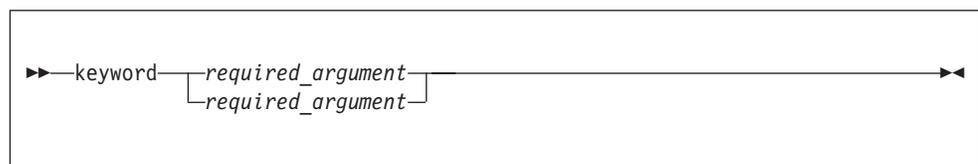


- Optional items are shown below the main path:

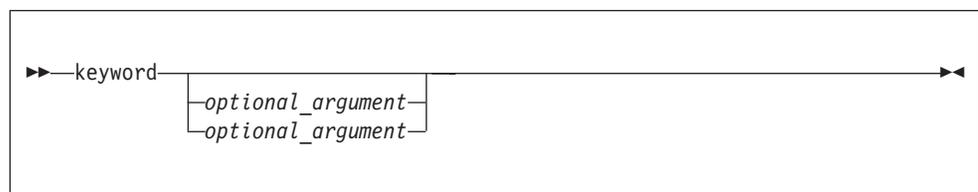


Note: Optional items (not in syntax diagrams) are enclosed by square brackets ([and]). For example, [UNIT=]u

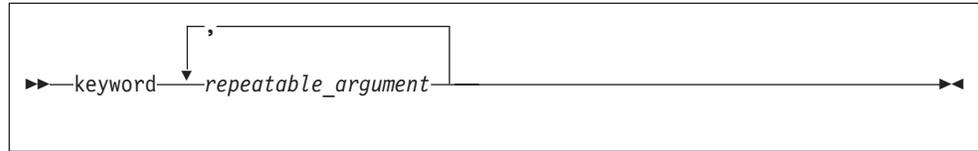
- If you can choose from two or more items, they are shown vertically, in a stack. If you *must* choose one of the items, one item of the stack is shown on the main path:



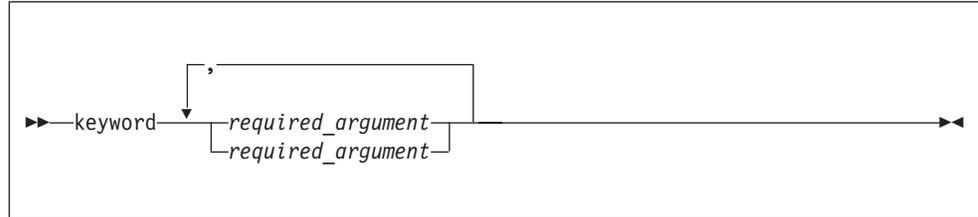
If choosing one of the items is optional, the entire stack is shown below the main path:



- An arrow returning to the left above the main line (a repeat arrow) indicates that you can repeat an item, and the separator character if it is other than a blank:

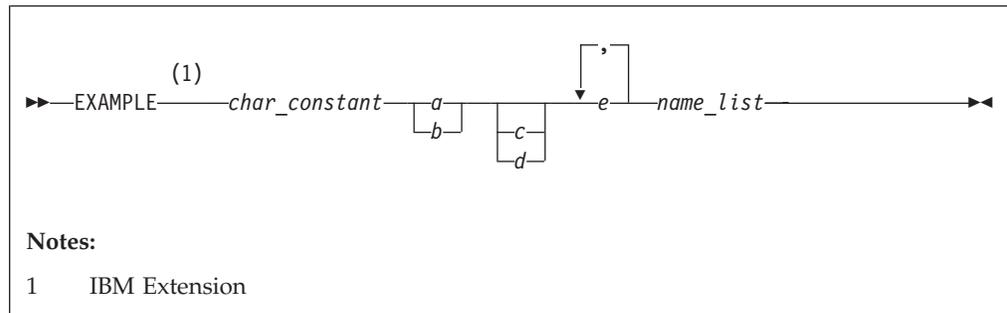


A repeat arrow above a stack indicates that you can make more than one choice from the items in the stack.



Sample syntax diagram

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



Interpret the diagram as follows:

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

How to read syntax statements

Syntax statements are read from left to right:

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

Example of a syntax statement

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

The following list explains the syntax statement:

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

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

Examples

The examples in this document are coded in a simple style that does not try to conserve storage, check for errors, achieve fast performance, or demonstrate recommended practice.

The examples in this document use the `xlf90`, `xlf90_r`, `xlf90_r7`, `xlf95`, `xlf95_r`, `xlf95_r7`, `xlf`, `xlf_r`, `xlf_r7`, `f77`, `fort77`, `f90`, and `f95` compiler invocation commands interchangeably. For more substantial source files, one of these commands may be more suitable than the others, as explained in “Compiling XL Fortran programs” on page 20.

Some sample programs from this document and some other programs that illustrate ideas presented in this document are in the directory `/usr/lpp/xlf/samples`.

Notes on the terminology used

Some of the terminology in this document is shortened, as follows:

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

Related information

IBM XL Fortran documentation

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 `/usr/lpp/xlf/` 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 V10.1 for AIX 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 information center are provided in the *XL Fortran Installation Guide*. The information center is also viewable on the Web at:

<http://publib.boulder.ibm.com/infocenter/comphelp/index.jsp>

- PDF documents

PDF documents are located by default in the `/usr/lpp/xlf/doc/language/pdf` directory, where *language* can be one of the following supported languages:

- en_US (U.S. English)
- ja_JP (Japanese)

PDF documents are also available on the Web at:

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

In addition to this document, the following files comprise the full set of XL Fortran product manuals:

Table 2. XL Fortran PDF files

Document title	PDF file name	Description
<i>IBM XL Fortran Enterprise Edition V10.1 for AIX Installation Guide</i>	install.pdf	Contains information for installing XL Fortran and configuring your environment for basic compilation and program execution.
<i>Getting Started with XL Fortran Enterprise Edition V10.1 for AIX</i>	getstart.pdf	Contains an introduction to the XL Fortran product, with overview information on setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
<i>IBM XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference</i>	cr.pdf	Contains information on setting up and configuring your compilation environment, compiling, linking and running programs, troubleshooting compilation errors, and descriptions of the various compiler options.
<i>IBM XL Fortran Enterprise Edition V10.1 for AIX Language Reference</i>	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.
<i>IBM XL Fortran Enterprise Edition V10.1 for AIX Optimization and Programming Guide</i>	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.

These PDF files are viewable and printable from Adobe Reader. If you do not have the Adobe Reader installed, you can download it from <http://www.adobe.com>

Additional documentation

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

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

Related documentation

You might want to consult the following publications, which are also referenced throughout this document:

- *AIX Commands Reference, Volumes 1 - 6*
- *AIX Technical Reference: Base Operating System and Extensions, Volumes 1 & 2*
- *OpenMP Application Program Interface Version 2.5*, available at <http://www.openmp.org>
- *ESSL for AIX on POWER V4.2 Guide and Reference*

Standards documents

XL Fortran is designed according to the following standards. You can refer to these standards for precise definitions of some of the features found in this document.

- *American National Standard Programming Language FORTRAN*, ANSI X3.9-1978.
- *American National Standard Programming Language Fortran 90*, ANSI X3.198-1992. (This document uses its informal name, Fortran 90.)
- *ANSI/IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985.
- *Federal (USA) Information Processing Standards Publication Fortran*, FIPS PUB 69-1.
- *Information technology - Programming languages - Fortran*, ISO/IEC 1539-1:1991 (E).
- *Information technology - Programming languages - Fortran - Part 1: Base language*, ISO/IEC 1539-1:1997. (This document uses its informal name, Fortran 95.)
- *Information technology - Programming languages - Fortran - Floating-Point Exception Handling*, ISO/IEC JTC1/SC22/WG5 N1379.
- *Information technology - Programming languages - Fortran - Enhance Data Type Facilities*, ISO/IEC JTC1/SC22/WG5 N1378.
- *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 2.5, (May 2005) specification*.

Technical support

Additional technical support is available from the XL Fortran Support page. This page provides a portal with search capabilities to a large selection of technical support FAQs and other support documents. You can find the XL Fortran Support page on the Web at:

<http://www.ibm.com/software/awdtools/fortran/xlfortran/support>

If you cannot find what you need, you can e-mail:

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. Introduction

This document describes IBM XL Fortran Enterprise Edition V10.1 for AIX and explains how to compile, link, and run programs that are written in the Fortran language.

You can use XL Fortran to compile XL Fortran program source files into executable object modules.

IBM XL Fortran Enterprise Edition V10.1 for AIX is an optimizing, standards-based, command-line compiler for the AIX operating system, running on PowerPC[®] hardware with the PowerPC architecture. The XL Fortran compiler enables application developers to create and maintain optimized 32-bit and 64-bit applications for the AIX operating system. The compiler also offers a diversified portfolio of optimization techniques that allow an application developer to exploit the multi-layered architecture of the PowerPC processor.

The implementation of the Fortran programming language is intended to promote portability among different environments by enforcing conformance to language standards. A program that conforms strictly to its language specification will have maximum portability among different environments. In theory, a program that compiles correctly with one standards-conforming compiler will compile and execute correctly under all other conforming compilers, insofar as hardware differences permit. A program that correctly exploits the extensions to the programming language in which it is written can improve the efficiency of its object code.

XL Fortran Enterprise Edition V10.1 for AIX can be used for large, complex, and computationally intensive programs. It also supports interlanguage calls with C/C++. For applications that require SIMD (single-instruction, multiple data) parallel processing, performance improvements can be achieved through optimization techniques, which may be less labor-intensive than vector programming. Many of the optimizations developed by IBM are controlled by compiler options and directives.

Chapter 2. Overview of XL Fortran features

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

Hardware and operating-system support

The XL Fortran Enterprise Edition Version 10.1 compiler is supported on the Version 5.1, or higher, AIX operating system. See the *IBM XL Fortran Enterprise Edition V10.1 for AIX Installation Guide* and README file for a list of requirements.

The compiler, its generated object programs, and run-time library will run on all RISC System/6000[®] (RS/6000[®]) or pSeries[®] systems with the required software, disk space, and virtual storage.

The POWER3, POWER4, POWER5, or POWER5+[™] processor is a type of PowerPC. In this document, any statement or reference to the PowerPC also applies to the POWER3, POWER4, POWER5, or POWER5+ processor.

To take maximum advantage of different hardware configurations, the compiler provides a number of options for performance tuning based on the configuration of the machine used for executing an application.

Language support

The XL Fortran language consists of the following:

- The full American National Standard Fortran 90 language (referred to as Fortran 90 or F90), defined in the documents *American National Standard Programming Language Fortran 90*, ANSI X3.198-1992 and *Information technology - Programming languages - Fortran*, ISO/IEC 1539-1:1991(E). This language has a superset of the features found in the FORTRAN 77 standard. It adds many more features that are intended to shift more of the tasks of error checking, array processing, memory allocation, and so on from the programmer to the compiler.
- The full ISO Fortran 95 language standard (referred to as Fortran 95 or F95), defined in the document *Information technology - Programming languages - Fortran - Part 1: Base language*, ISO/IEC 1539-1:1997.
- Extensions to the Fortran 95 standard:
 - Industry extensions that are found in Fortran products from various compiler vendors
 - Extensions specified in SAA Fortran
- Partial support of the Fortran 2003 standard

In the *XL Fortran Language Reference*, extensions to the Fortran 95 language are marked as described in the *Typographical Conventions* topic.

Migration support

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

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

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

To protect your investments in FORTRAN 77 object code, you can link Fortran 90 and Fortran 95 programs with existing FORTRAN 77 object modules and libraries. See “Linking new objects with existing ones” on page 35 for details.

Source-code conformance checking

To help you find anything in your programs that might cause problems when you port to or from different Fortran 2003, FORTRAN 77, Fortran 90, or Fortran 95 compilers, the XL Fortran compiler provides options that warn you about features that do not conform to certain Fortran definitions.

If you specify the appropriate compiler options, the XL Fortran compiler checks source statements for conformance to the following Fortran language definitions:

- Full Fortran 2003 Standard (`-qlanglvl=2003std` option), full American National Standard FORTRAN 77 (`-qlanglvl=77std` option), full American National Standard Fortran 90 (`-qlanglvl=90std` option), and full Fortran 95 standard (`-qlanglvl=95std` option)
- Fortran 90, less any obsolescent features (`-qlanglvl=90pure` option)
- Fortran 95, less any obsolescent features (`-qlanglvl=95pure` option)
- Fortran 2003, less any obsolescent features (`-qlanglvl=2003pure` option)
- IBM SAA[®] FORTRAN (`-qsaa` option)

You can also use the `langlvl` environment variable for conformance checking.

Note: Fortran 2003 conformance checking is based on XL Fortran’s current, subset implementation of this standard.

Highly configurable compiler

You can invoke the compiler by using the `xlf`, `xlf_r`, `xlf_r7`, `xlf90`, `xlf90_r`, `xlf90_r7`, `xlf95`, `xlf95_r`, `xlf95_r7`, `f77`, or `fort77` command. The `xlf`, `xlf_r`, `xlf_r7`, and `f77` commands maintain maximum compatibility with the behavior and I/O formats of XL Fortran Version 2. The `f90`, `xlf90`, `xlf90_r`, and `xlf90_r7` commands provide more Fortran 90 conformance and some implementation choices for efficiency and

usability. The **f95**, **xl95**, **xl95_r**, and **xl95_r7** commands provide more Fortran 95 conformance and some implementation choices for efficiency and usability. The **f77** or **fort77** command provides maximum compatibility with the XPG4 behavior.

The main difference between the set of **xl_r**, **xl90_r**, **xl90_r7**, **xl95_r**, and **xl95_r7** commands and the set of **xl**, **xl90**, **xl95**, **f77**, **fort77**, **f90**, and **f95** commands is that the first set links and binds the object files to the thread-safe components (libraries, `crt0_r.o`, and so on). You can have this behavior with the second set of commands by using the **-F** compiler option to specify the configuration file stanza to use. For example:

```
xl -F/etc/xl.cfg:xl_r
```

You can control the actions of the compiler through a set of options. The different categories of options help you to debug, to optimize and tune program performance, to select extensions for compatibility with programs from other platforms, and to do other common tasks that would otherwise require changing the source code.

To simplify the task of managing many different sets of compiler options, you can customize the single file `/etc/xl.cfg` instead of creating many separate aliases or shell scripts.

Related information:

- “Customizing the configuration file” on page 11
- “Compiling XL Fortran programs” on page 20
- “Summary of the XL Fortran compiler options” on page 51 and “Detailed descriptions of the XL Fortran compiler options” on page 76
- “Understanding XL Fortran messages” on page 276

Diagnostic listings

The compiler output listing has optional sections that you can include or omit. For information about the applicable compiler options and the listing itself, refer to “Options that control listings and messages” on page 62 and Chapter 8, “Understanding XL Fortran compiler listings,” on page 295.

The **-S** option gives you a true assembler source file.

Symbolic debugger support

You can use **dbx** and other symbolic debuggers for your programs.

Program optimization

The XL Fortran compiler helps you control the optimization of your programs:

- You can select different levels of compiler optimizations.
- You can turn on separate optimizations for loops, floating point, and other categories.
- You can optimize a program for a particular class of machines or for a very specific machine configuration, depending on where the program will run.

The *XL Fortran Optimization and Programming Guide* provides a road map and optimization strategies for these features.

Chapter 3. Setting up and customizing XL Fortran

This section explains how to customize XL Fortran settings for yourself or all users and how to set up a user account to use XL Fortran. The full installation procedure is beyond the scope of this section, which refers you to the documents that cover the procedure in detail.

This section can also help you to diagnose problems that relate to installing or configuring the compiler.

Some of the instructions require you to be a superuser, and so they are only applicable if you are a system administrator.

Where to find installation instructions

To install the compiler, refer to these documents (preferably in this order):

1. Read the file called `/usr/lpp/xlf/doc/en_US/README`, and follow any directions it gives. It contains information that you should know and possibly distribute to other people who use XL Fortran.
2. Read the *XL Fortran Installation Guide* to see if there are any important notices you should be aware of or any updates you might need to apply to your system before doing the installation.
3. Read the *AIX Installation Guide* to understand the full procedure for installing optional software from the distribution medium. This document provides good step-by-step instructions that should answer all your questions and should help you to solve any installation problems.

If you are already experienced with AIX software installation, you can use the `installp` or `smit installp` command to install all the images from the distribution medium.

Using the compiler on a network file System

If you want to use the XL Fortran compiler on a Network File System server for a networked cluster of machines, use the Network Install Manager.

The following directories under `/usr` contain XL Fortran components:

- `/usr/bin` contains the compiler invocation commands.
- `/usr/lib` contains the libraries.
- `/usr/lpp/xlf` contains executables and files that the compiler needs.
- `/usr/include` contains the include files, some of which contain definitions that XL Fortran uses.
- `/usr/lib/nls/msg` contains the message catalog files that XL Fortran uses.
- `/usr/lpp/xlfrtemsg` contains the default message catalog files that are used by XL Fortran programs.
- `/usr/share/man/cat1` contains the compiler man pages.
- `/usr/share/man/info/en_US/xlf/pdf` contains the PDF format of the English XL Fortran publications.
- `/usr/share/man/info/en_US/xlf/html` contains the HTML format of the English XL Fortran publications.

You must also copy the `/etc/xlf.cfg` file from the server to the client. The `/etc` directory contains the configuration files specific to a machine, and it should not be mounted from the server.

Correct settings for environment variables

You can set and export a number of environment variables for use with the operating system. The following sections deal with the environment variables that have special significance to the XL Fortran compiler, application programs, or both.

Environment variable basics

You can set the environment variables from shell command lines or from within shell scripts. If you are not sure which shell is in use, a quick way to find out is to issue an `echo $0`. This provides a different result in each shell:

```
$ sh
$ echo $0
sh
$ ksh
$ echo $0
ksh
$ csh
% echo $0
No file for $0.
%
```

The Bourne shell path is `/bin/sh`, the Korn shell path is `/bin/ksh`, and the C shell path is `/bin/csh`.

To set the environment variables so that everyone on the system has access to them, set the variables in the file `/etc/profile` (for the Bourne or the Korn shell), or set the variables in the file `/etc/csh.login` or in the file `/etc/csh.cshrc` (for the C shell). To set them for a specific user only, add the appropriate commands to the appropriate `.profile` or `.cshrc` file in the user's home directory. The variables are set the next time the user logs on.

For more information about setting environment variables, see the *AIX Commands Reference*.

The following examples show how to set environment variables from various shells.

From the Bourne or Korn shell:

```
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/prime/%N
LANG=en_US
TMPDIR=/home/joe/temp
export LANG NLSPATH TMPDIR
```

From the C shell:

```
setenv LANG en_US
setenv NLSPATH /usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/prime/%N
setenv TMPDIR /home/joe/temp
```

To display the contents of an environment variable, enter the command `echo $var_name`.

Note: For the remainder of this document, most examples of shell commands use `ksh` notation instead of repeating the syntax for all shells.

Environment variables for national language support

Diagnostic messages and the listings from the compiler appear in the default language that was specified at installation of the operating system. If you want the messages and listings to appear in another language, you can set and export the following environment variables before executing the compiler:

LANG	Specifies the <i>locale</i> . A locale is divided into categories. Each category contains a specific aspect of the locale data. Setting LANG may change the national language for all the categories.
NLSPATH	Refers to a list of directory names where the message catalogs may be found.

For example, to specify the Japanese locale with the **IBM_eucJP** code page, use the following commands from the Bourne or Korn shell:

```
LANG=ja_JP
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/prime/%N
export LANG NLSPATH
```

Substitute any valid national language code for **ja_JP**, provided the associated message catalogs are installed.

These environment variables are initialized when the operating system is installed and may be different from the ones that you want to use with the compiler.

Each category has an environment variable associated with it. If you want to change the national language for a specific category but not for other categories, you can set and export the corresponding environment variable.

For example:

LC_MESSAGES

Specifies the national language for the messages that are issued. It affects messages from the compiler and XLF-compiled programs, which may be displayed on the screen or stored in a listing, module, or other compiler output file.

LC_TIME

Specifies the national language for the time format category. It primarily affects the compiler listings.

LC_CTYPE

Defines character classification, case conversion, and other character attributes. For XL Fortran, it primarily affects the processing of multibyte characters.

LC_NUMERIC

Specifies the format to use for input and output of numeric values. Setting this variable in the shell *does not* affect either the compiler or XLF-compiled programs. The first I/O statement in a program sets the **LC_NUMERIC** category to **POSIX**. Therefore, programs that require a different setting must reset it after this point and should restore the setting to **POSIX** for all I/O statements.

Notes:

1. Specifying the **LC_ALL** environment variable overrides the value of the **LANG** and other **LC_** environment variables.
2. If the XL Fortran compiler or application programs cannot access the message catalogs or retrieve a specific message, the message appears in U.S. English.

3. The backslash character, \, has the same hexadecimal code, X'5C', as the Yen symbol and can appear as the Yen symbol on the display if the locale is Japanese.

Related information: “Selecting the language for run-time messages” on page 40.

See *General Programming Concepts: Writing and Debugging Programs* for more information about National Language Support environment variables and locale concepts.

LIBPATH:Setting library search paths

Under normal circumstances, you only need **LIBPATH** if libraries are located in different directories at run time from those that they are in at compile time. To use **LIBPATH**, set it at run time to the names of any directories that contain required user libraries, plus **/usr/lib**:

```
# Compile and link
xlf95 -L/usr/lib/mydir1 -L/usr/lib/mydir2 -lmylib1 -lmylib2 test.f

# When the libraries are in the same directories as at compile
# time, the program finds them.
a.out

# If libmylib1.a and libmylib2.a are moved to /usr/lib/mydir3,
# you must set the LIBPATH variable:
export LIBPATH=/usr/lib/mydir3:/usr/lib
a.out
```

When running the compiler, ensure that the library **libxlf90.a** is in **/usr/lib** or is in a directory named in the **LIBPATH** setting. Otherwise, you cannot run the compiler, because it is dynamically linked with the **libxlf90.a** library.

PDFDIR: Specifying the directory for PDF profile information

When you compile a Fortran program with the **-qpdf** compiler option, you can specify the directory where profiling information is stored by setting the **PDFDIR** environment variable to the name of the directory. The compiler creates the files to hold the profile information. XL Fortran updates the files when you run an application that is compiled with the **-qpdf1** option.

Because problems can occur if the profiling information is stored in the wrong place or is updated by more than one application, you should follow these guidelines:

- Always set the **PDFDIR** variable when using the **-qpdf** option.
- Store the profiling information for each application in a different directory, or use the **-qipa=pdfname=[filename]** option to explicitly name the temporary profiling files according to the template provided.
- Leave the value of the **PDFDIR** variable the same until you have completed the PDF process (compiling, running, and compiling again) for the application.

TMPDIR: Specifying a directory for temporary files

The XL Fortran compiler creates a number of temporary files for use during compilation. An XL Fortran application program creates a temporary file at run time for a file opened with **STATUS='SCRATCH'**. By default, these files are placed in the directory **/tmp**.

If you want to change the directory where these files are placed, perhaps because `/tmp` is not large enough to hold all the temporary files, set and export the `TMPDIR` environment variable before running the compiler or the application program.

If you explicitly name a scratch file by using the `XLFSCRATCH_unit` method described below, the `TMPDIR` environment variable has no effect on that file.

XLFSCRATCH_unit: Specifying names for scratch files

To give a specific name to a scratch file, you can set the run-time option `scratch_vars=yes`; then set one or more environment variables with names of the form `XLFSCRATCH_unit` to file names to use when those units are opened as scratch files. See *Naming scratch files* in the *XL Fortran Optimization and Programming Guide* for examples.

XLFUNIT_unit: Specifying names for implicitly connected files

To give a specific name to an implicitly connected file or a file opened with no `FILE=` specifier, you can set the run-time option `unit_vars=yes`; then set one or more environment variables with names of the form `XLFUNIT_unit` to file names. See *Naming files that are connected with no explicit name* in the *XL Fortran Optimization and Programming Guide* for examples.

Customizing the configuration file

The configuration file specifies information that the compiler uses when you invoke it. XL Fortran provides the default configuration file `/etc/xf.cfg` at installation time.

If you are running on a single-user system, or if you already have a compilation environment with compilation scripts or makefiles, you may want to leave the default configuration file as it is.

Otherwise, especially if you want many users to be able to choose among several sets of compiler options, you may want to add new named stanzas to the configuration file and to create new commands that are links to existing commands. For example, you could specify something similar to the following to create a link to the `xf95` command:

```
ln -s /bin/xf95 /home/username/bin/my_xf95
```

When you run the compiler under another name, it uses whatever options, libraries, and so on, that are listed in the corresponding stanza.

Notes:

1. The configuration file contains other named stanzas to which you may want to link.
2. If you make any changes to the default configuration file and then move or copy your makefiles to another system, you will also need to copy the changed configuration file.
3. Installing a compiler program temporary fix (PTF) or an upgrade may overwrite the `/etc/xf.cfg` file. Therefore, be sure to save a copy of any modifications you have made before doing such an installation.
4. If you upgrade the operating system, you must change the symbolic link in `/etc/xf.cfg` to point to the correct version of the configuration file.

5. You cannot use tabs as separator characters in the configuration file. If you modify the configuration file, make sure that you use spaces for any indentation.
6. The `xlf_r`, `xlf90_r`, and `xlf95_r` stanzas support the operating system default thread interface: that is, the POSIX1003.1-1996 standard for AIX Version 5.1 and higher levels of the AIX operating system.
7. If you are mixing Message-Passing Interface (MPI) and threaded programming, use the appropriate stanza in the `xlf.cfg` file to link in the proper libraries and to set the correct default behavior:

mpxlf	Specifies <code>xlf</code> and <code>f77</code> behavior when using MPI.
mpxlf_r	Specifies <code>xlf_r</code> behavior when mixing MPI and threaded programming. You access this stanza by specifying the mpxlf_r command.
mpxlf_r7	Specifies <code>xlf_r7</code> behavior when mixing MPI and threaded programming. You access this stanza by specifying the mpxlf_r command with the -d7 option. The level of POSIX pthreads API support is Draft 7.
mpxlf90	Specifies <code>xlf90</code> behavior when using MPI.
mpxlf90_r	Specifies <code>xlf90_r</code> behavior when mixing MPI and threaded programming. You access this stanza by specifying the mpxlf90_r command.
mpxlf90_r7	Specifies <code>xlf90_r7</code> behavior when mixing MPI and threaded programming. You access this stanza by specifying the mpxlf90_r command with the -d7 option. The level of POSIX pthreads API support is Draft 7.
mpxlf95	Specifies <code>xlf95</code> behavior when using MPI.
mpxlf95_r	Specifies <code>xlf95_r</code> behavior when mixing MPI and threaded programming. You access this stanza by specifying the mpxlf95_r command.
mpxlf95_r7	Specifies <code>xlf95_r7</code> behavior when mixing MPI and threaded programming. You access this stanza by specifying the mpxlf95_r command with the -d7 option. The level of POSIX pthreads API support is Draft 7.

Attributes

The configuration file contains the following attributes:

- | | |
|---------------|---|
| use | The named and local stanzas provide the values for attributes. For single-valued attributes, values in the use attribute apply if there is no value in the local, or default, stanza. For comma-separated lists, the values from the use attribute are added to the values from the local stanza. You can only use a single level of the use attribute. Do not specify a use attribute that names a stanza with another use attribute. |
| crt | When invoked in 32-bit mode, the default (which is the path name of the object file that contains the startup code), passed as the first parameter to the linkage editor. |
| crt_64 | When invoked in 64-bit mode, using -q64 for example, the path name of the object file that contains the startup code, passed as the first parameter to the linkage editor. |
| mcrt | Same as for crt , but the object file contains profiling code for the -p option. |

mcrt_64	Same as for crt_64 , but the object file contains profiling code for the -p option.
gcrt	Same as crt , but the object file contains profiling code for the -pg option.
gcrt_64	Same as crt_64 , but the object file contains profiling code for the -pg option.
cpp	The absolute path name of the C preprocessor, which is automatically called for files ending with a specific suffix (usually .F).
xlf	The absolute path name of the main compiler executable file. The compiler commands are driver programs that execute this file.
code	The absolute path name of the optimizing code generator.
xlfopt	Lists names of options that are assumed to be compiler options, for cases where, for example, a compiler option and a linker option use the same letter. The list is a concatenated set of single-letter flags. Any flag that takes an argument is followed by a colon, and the whole list is enclosed by double quotation marks.
as	The absolute path name of the assembler.
asopt	Lists names of options that are assumed to be assembler options for cases where, for example, a compiler option and an assembler option use the same letter. The list is a concatenated set of single-letter flags. Any flag that takes an argument is followed by a colon, and the whole list is enclosed by double quotation marks. You may find it more convenient to set up this attribute than to pass options to the assembler through the -W compiler option.
ld	The absolute path name of the linker.
ldopt	Lists names of options that are assumed to be linker options for cases where, for example, a compiler option and a linker option use the same letter. The list is a concatenated set of single-letter flags. Any flag that takes an argument is followed by a colon, and the whole list is enclosed by double quotation marks. You may find it more convenient to set up this attribute than to pass options to the linker through the -W compiler option. However, most unrecognized options are passed to the linker anyway.
options	A string of options that are separated by commas. The compiler processes these options as if you entered them on the command line before any other option. This attribute lets you shorten the command line by including commonly used options in one central place.
cppoptions	A string of options that are separated by commas, to be processed by cpp (the C preprocessor) as if you entered them on the command line before any other option. This attribute is needed because some cpp options are usually required to produce output that can be compiled by XL Fortran. The default option is -C , which preserves any C-style comments in the output.
fsuffix	The allowed suffix for Fortran source files. The default is f . The compiler requires that all source files in a single compilation have the same suffix. Therefore, to compile files with other suffixes, such

as **f95**, you must change this attribute in the configuration file or use the **-qsuffix** compiler option. For more information on **-qsuffix**, see “-qsuffix option” on page 233.

cppsuffix	The suffix that indicates a file must be preprocessed by the C preprocessor (cpp) before being compiled by XL Fortran. The default is F .
osuffix	The suffix used to recognize object files that are specified as input files. The default is o .
ssuffix	The suffix used to recognize assembler files that are specified as input files. The default is s .
libraries	-l options, which are separated by commas, that specify the libraries used to link all programs.
proflibs	-L options, which are separated by commas, that specify the path where the linker searches for additional libraries for profiled programs.
smplibraries	Specifies the libraries that are used to link programs that you compiled with the -qsmpl compiler option.
hot	Absolute path name of the program that does array language transformations.
ipa	Absolute path name of the program that performs interprocedural optimizations, loop optimizations, and program parallelization.
bolt	Absolute path name of the binder.
defaultmsg	Absolute path name of the default message files.
include	Indicates the search path that is used for compilation include files and module files.
include_32	Indicates the search path that is used for 32-bit compilation include files.
include_64	Indicates the search path that is used for 64-bit compilation include files.

Note: To specify multiple search paths for compilation include files, separate each path location with a comma as follows:

```
include = -l/path1, -l/path2, ...
```

Related information: You can use the “-F option” on page 93 to select a different configuration file, a specific stanza in the configuration file, or both.

Determining which level of XL Fortran is installed

Sometimes, you may not be sure which level of XL Fortran is installed on a particular machine. You would need to know this information before contacting software support.

To check whether the latest level of the product has been installed through the system installation procedure, issue the command:

```
ls1pp -h "*xlf*"
```

The result includes the version, release, modification, and fix level of the compiler image installed on the system.

To check the level of the compiler executable itself, issue the command:

```
what /usr/lpp/xlf/bin/xlfentry
```

If the compiler is installed in a different directory, use the appropriate path name for the `xlfentry` file.

You can also use the `-qversion` compiler option to see the version and release for the compiler.

Upgrading to XL Fortran Version 10

Here is some advice to help make the transition from an earlier version of the XL Fortran compiler as fast and simple as possible.

Things to note in XL Fortran Version 10

Because XL Fortran Version 10.1 is highly compatible with XL Fortran Versions 9 through 3 inclusive, most of the advice in this section applies to upgrades from Version 2, or earlier levels of XL Fortran.

- The `f90`, `xlf90`, `xlf90_r`, and `xlf90_r7` commands provide Fortran 90 conformance, and the `f95`, `xlf95`, `xlf95_r`, and `xlf95_r7` commands provide Fortran 95 conformance. However, these commands may cause some problems with existing FORTRAN 77 programs. The `xlf`, `xlf_r`, `xlf_r7`, `f77`, and `fort77` commands avoid some of these problems by keeping the old behavior wherever possible.
- Fortran 90 introduced the idea of kind parameters for types. Except for the types complex and character, XL Fortran uses numeric kind parameters that correspond to the lengths of the types. For the type complex, the kind parameter is equal to the length of the real portion, which is half of the overall length. For the type character, the kind parameter is equal to the number of bytes that are required to represent each character, and this value is 1. A FORTRAN 77 declaration that is written using the * extension for length specifiers can now be rewritten with a kind parameter:

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

This new form is the one we use consistently throughout the XL Fortran manuals.

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

- Fortran 90 introduced a standardized free source form for source code, which is different from the XL Fortran Version 2 free source form. The `-qfree` and `-k` options now use the Fortran 90 free source form; the Version 2 free source form is available through the option `-qfree=ibm`.
- The library that provides Fortran 2003, Fortran 90, and Fortran 95 support is `libxlf90.a`, located in `/usr/lib`. A `libxlf.a` library of stub routines is provided in `/usr/lib`, but it is only used for linking existing Version 1 or 2 object files or

running existing executables. When a Version 1 or Version 2 object file calls entry points in `libxlf.a`, those entry points then call equivalent entry points in `libxlf90.a`. If you recompile such object files, the result could be improved I/O performance, because the entry points in `libxlf90.a` are called directly.

Avoiding or fixing upgrade problems

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

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

Note that in the following table, you can substitute `xlf_r` or `xlf_r7` for `xlf`, `xlf90_r` or `xlf90_r7` for `xlf90`, and `xlf95_r` or `xlf95_r7` for `xlf95`.

Table 3. Potential problems migrating programs to XL Fortran Version 10. The column on the right shows which problems you can avoid by using the `xlf` or `f77` command.

Potential Problem	Solution or Workaround	xlf Avoids?
Compilation Problems		
New intrinsic procedure names may conflict with external procedure names. The intrinsic procedure is called instead of the external procedure.	Use the <code>-qextern</code> option, or insert <code>EXTERNAL</code> statements to avoid the ambiguity. Consider switching to the Fortran 90 or Fortran 95 procedure if it does what you want.	
The <code>.XOR.</code> intrinsic operator is not recognized.	Use the option <code>-qxlf77=intxor.</code>	✓
Zero-sized objects are not allowed by the compiler.	Use the <code>xlf90</code> or <code>xlf95</code> command, or use the <code>-qzerosize</code> option with the <code>xlf</code> or <code>f77</code> command.	
Performance / Optimization Problems		
Existing programs or programs linked with older XL Fortran object files run more slowly or do not show expected performance improvements on new hardware.	Recompile everything.	
Programs compiled with <code>-O3</code> or <code>-qhot</code> optimization behave differently from those unoptimized (different results, exceptions, or compilation messages).	Try adding the <code>-qstrict</code> option.	
The option combination <code>-O</code> and <code>-1</code> cannot be abbreviated <code>-O1</code> , to avoid misunderstandings. (There are <code>-O2</code> , <code>-O3</code> , <code>-O4</code> , and <code>-O5</code> optimization levels, but there is no <code>-O1</code> .)	Specify <code>-O</code> and <code>-1</code> as separate options.	

Table 3. Potential problems migrating programs to XL Fortran Version 10 (continued). The column on the right shows which problems you can avoid by using the **xlf** or **f77** command.

Potential Problem	Solution or Workaround	xlf Avoids?
Programs that use integer POINTERS produce incorrect results when optimized.	Specify the option -qalias=intptr with the xlf90 or xlf95 command, or use the xlf command.	✓
Run-Time Problems		
Programs that read to the end of the file and then try to append records without first executing a BACKSPACE statement do not work correctly. The write requests generate error messages.	To compile existing programs, specify the option -qxlf77=softeof with the xlf90 or xlf95 command, or use the xlf command. For new programs, add the BACKSPACE statement before writing past the endfile record.	✓
Uninitialized variables are not necessarily set to zero, and programs that ran before may exceed the user stack limit. The reason is that the default storage class is now AUTOMATIC , rather than STATIC (an implementation choice allowed by the language).	Ensure that you explicitly initialize your variables, use the -qsave option with the xlf90 or xlf95 command, or add SAVE statements where needed in the source.	✓
Writing data to some files opened without a POSITION= specifier overwrites the files, instead of appending the data.	Use the option -qposition=appendold , or add POSITION= specifiers where needed.	✓
Newly compiled programs are unable to read existing data files containing NAMELIST data. The reason is that the Fortran 90 and Fortran 95 standards define a namelist format that is different from that used on AIX in the past.	Set the environment variable XLFRTEOPTS to the string namelist=old . The programs that produced the old NAMELIST data must be recompiled.	
Some I/O statements and edit descriptors accept or produce slightly different input and output. For example, real output now has a leading zero when appropriate. The changes to I/O formats are intended to be more usable and typical of industry practice, so you should try to use the defaults for any new data you produce.	When you need to maintain compatibility with existing data files, compile with the xlf command. If the incompatibility is due to a single specific I/O change, see if the -qxlf77 option has a suboption for backward compatibility. If so, you can switch to the xlf90 or xlf95 command and use the -qxlf77 option on programs that use the old data files.	✓

Table 3. Potential problems migrating programs to XL Fortran Version 10 (continued). The column on the right shows which problems you can avoid by using the `xlf` or `f77` command.

Potential Problem	Solution or Workaround	xlf Avoids?
Numeric results and I/O output are not always exactly identical with XL Fortran Version 2. Certain implementation details of I/O, such as spacing in list-directed output and the meanings of some <code>IOSTAT</code> values, have changed since XL Fortran Version 2. (This entry is similar to the previous one except that these differences have no backward-compatibility switches.)	You may need to generate existing data files again or to change any programs that depend on these details. When no backward-compatibility switch is provided by the <code>-qxlf77</code> compiler option or <code>XLFRTEOPTS</code> run-time options, there is no way to get the old behavior back.	
<code>SIGN(A,B)</code> now returns <code>- A </code> when <code>B = -0.0</code> . Prior to XL Fortran Version 7.1, it returned <code> A </code> .	This behavior conforms with the Fortran 95 standard and is consistent with the IEEE standard for binary floating-point arithmetic. It occurs because the <code>-qxlf90=signedzero</code> option is turned on. Turn it off, or specify a command that does not use this option by default.	✓
A minus sign is printed for a negative zero in formatted output. A minus sign is printed for negative values that have an outputted form of zero (that is, in the case where trailing non-zero digits are truncated from the output so that the resulting output looks like zero). Prior to XL Fortran Version 7.1, minus signs were not printed in these situations.	This behavior conforms with the Fortran 95 standard and occurs because the <code>-qxlf90=signedzero</code> option is turned on. Turn it off, or specify a command that does not use this option by default.	✓

Related information:

- “Compiling for specific architectures” on page 28
- “Setting run-time options” on page 40
- “-qalias option” on page 109
- “-qextern option” on page 146
- “-qposition option” on page 207
- “-qsave option” on page 216
- “-qxlf77 option” on page 252

Running two levels of XL Fortran

It is possible for two different levels of the XL Fortran compiler to coexist on one system. This allows you to invoke one level by default and to invoke the other one whenever you explicitly choose to.

To do this, consult the *XL Fortran Installation Guide* for details.

Chapter 4. Editing, compiling, linking, and running XL Fortran programs

Most Fortran program development consists of a repeating cycle of editing, compiling and linking (which is by default a single step), and running. If you encounter problems at some part of this cycle, you may need to refer to the sections that follow this one for help with optimizing, debugging, and so on.

Prerequisite information:

1. Before you can use the compiler, all the required AIX settings (for example, certain environment variables and storage limits) must be correct for your user ID; for details, see “Correct settings for environment variables” on page 8.
2. Before using the compiler for a specialized purpose, such as porting or performance tuning, look at the categories of options in “Summary of the XL Fortran compiler options” on page 51 to see if XL Fortran already provides a solution.
3. To learn more about writing and optimizing XL Fortran programs, refer to the *XL Fortran Language Reference* and *XL Fortran Optimization and Programming Guide*.

libxlf90.a is provided for both threaded and non-threaded applications. XL Fortran determines at run time whether your application is threaded.

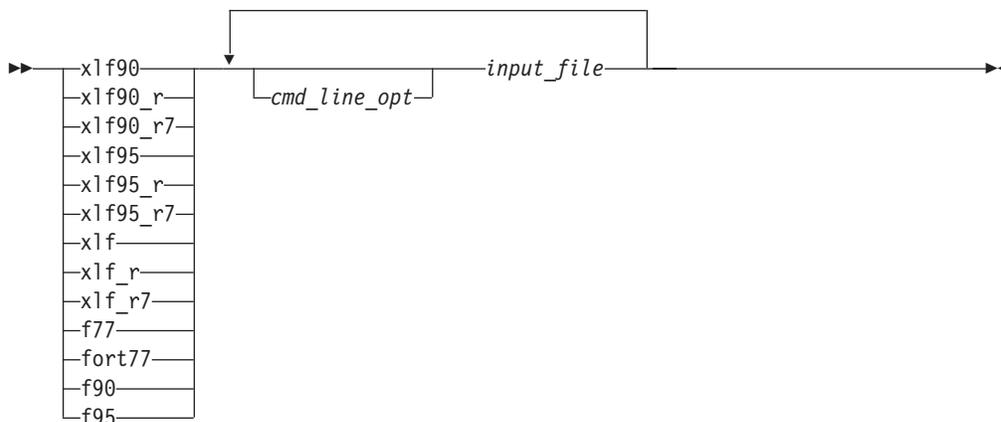
Editing XL Fortran source files

To create Fortran source programs, you can use any of the available text editors, such as **vi** or **emacs**. Source programs must have a suffix of **.f** unless the **fsuffix** attribute in the configuration file specifies a different suffix or the **-qsuffix** compiler option is used. You can also use a suffix of **.F** if the program contains C preprocessor (**cpp**) directives that must be processed before compilation begins. Source files with the **.f77**, **.f90**, or **.f95** suffix are also valid.

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

Compiling XL Fortran programs

To compile a source program, use one of the `xl f90`, `xl f90_r`, `xl f90_r7`, `xl f95`, `xl f95_r`, `xl f95_r7`, `xl f`, `xl f_r`, `xl f_r7`, `f77`, `fort77`, `f90`, or `f95` commands, which have the form:



These commands all accept essentially the same Fortran language. The main difference is that they use different default options (which you can see by reading the file `/etc/xlf.cfg`).

The invocation command performs the necessary steps to compile the Fortran source files, assemble any `.s` files, and link the object files and libraries into an executable program. In particular, the `xl f_r`, `xl f_r7`, `xl f90_r`, `xl f90_r7`, `xl f95_r`, and `xl f95_r7` commands use the thread-safe components (libraries, `crto_r.o`, and so on) to link and bind object files.

The following table summarizes the invocation commands that you can use:

Table 4. XL Fortran Invocation commands

Driver Invocation	Path or Location	Chief Functionality	Linked Libraries
<code>xl f90</code>	<code>/usr/bin</code>	Fortran 90	<code>libxlf90.a</code>
<code>xl f90_r</code>	<code>/usr/bin</code>	Threadsafe Fortran 90, operating system default POSIX pthreads API	<code>libxlf90.a</code>
<code>xl f90_r7</code>	<code>/usr/bin</code>	Threadsafe Fortran 90, Draft 7 POSIX pthreads API	<code>libxlf90.a</code>
<code>xl f95</code>	<code>/usr/bin</code>	Fortran 95	<code>libxlf90.a</code>
<code>xl f95_r</code>	<code>/usr/bin</code>	Threadsafe Fortran 95, operating system default POSIX pthreads API	<code>libxlf90.a</code>
<code>xl f95_r7</code>	<code>/usr/bin</code>	Threadsafe Fortran 95, Draft 7 POSIX pthreads API	<code>libxlf90.a</code>
<code>xl f</code>	<code>/usr/bin</code>	FORTTRAN 77	<code>libxlf90.a</code>
<code>xl f_r</code>	<code>/usr/bin</code>	Threadsafe FORTRAN 77, operating system default POSIX pthreads API	<code>libxlf90.a</code>

Table 4. XL Fortran Invocation commands (continued)

Driver Invocation	Path or Location	Chief Functionality	Linked Libraries
<code>xlf_r7</code>	<code>/usr/bin</code>	Threadsafe FORTRAN 77, Draft 7 POSIX pthreads API	<code>libxlf90.a</code>
<code>f77</code> or <code>fort77</code>	<code>/usr/bin</code>	FORTRAN 77	<code>libxlf90.a</code>
<code>f90</code>	<code>/usr/bin</code>	Fortran 90	<code>libxlf90.a</code>
<code>f95</code>	<code>/usr/bin</code>	Fortran 95	<code>libxlf90.a</code>

The invocation commands have the following implications for directive triggers:

- For `f77`, `fort77`, `f90`, `f95`, `xlf`, `xlf90`, and `xlf95`, the directive trigger is **IBM*** by default.
- For all other commands, the directive triggers are **IBM*** and **IBMT** by default. If you specify `-qsmp`, the compiler also recognizes the **IBMP**, **SMP\$**, and **\$OMP** trigger constants. If you specify the `-qsmp=omp` option, the compiler only recognizes the **\$OMP** trigger constant.

If you specify the `-qsmp` compiler option, the following occurs:

- The compiler turns on automatic parallelization.
- The compiler recognizes the **IBMP**, **IBMT**, **IBM***, **SMP\$**, and **\$OMP** directive triggers.

XL Fortran provides the library `libxlf90_t.a`, in addition to `libxlf90.a`.

It will be installed as `/usr/lib/libxlf90_t.a` with one restriction on its use: because routines in the library are not thread-reentrant, only one Fortran thread at a time can perform I/O operations or invoke Fortran intrinsics in a multithreaded application that uses the library. To avoid the thread synchronization overhead in `libxlf90.a` when an application is threaded, you can use `libxlf90_t.a` in multithreaded applications where there is only one Fortran thread.

When you bind a multi-threaded executable with multiple Fortran threads, `-lxlf90` should appear instead of `-lxlf90_t` in the command line. Note that using the `xlf_r`, `xlf_r7`, `xlf90_r`, `xlf90_r7`, `xlf95_r`, or `xlf95_r7` invocation command ensures the proper linking.

Compiling XL Fortran Version 2 programs

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

`f77` is identical to `xlf` (assuming that the configuration file has not been customized).

You may find that you need to continue using these commands for compatibility with existing makefiles and build environments. However, be aware that programs that you compile with these commands may not conform to the Fortran 90 or Fortran 95 standard in subtle ways.

Compiling Fortran 90 or Fortran 95 programs

The `f90`, `xlf90`, `xlf90_r`, and `xlf90_r7` commands make your programs conform more closely to the Fortran 90 standard than do the `xlf`, `xlf_r`, `xlf_r7`, and `f77/fort77`

commands. The **f95**, **xlF95**, **xlF95_r**, and **xlF95_r7** commands make your programs conform more closely to the Fortran 95 standard than do the **xlF**, **xlF_r**, **xlF_r7**, and **f77/fort77** commands. **f90**, **xlF90**, **xlF90_r**, **xlF90_r7**, **f95**, **xlF95**, **xlF95_r**, and **xlF95_r7** are the preferred commands for compiling any new programs. They all accept Fortran 90 free source form by default; to use them for fixed source form, you must use the **-qfixed** option. I/O formats are slightly different between these commands and the other commands. I/O formats also differ between the set of **f90**, **xlF90**, **xlF90_r**, and **xlF90_r7** commands and the set of **f95**, **xlF95**, **xlF95_r**, and **xlF95_r7** commands. We recommend that you switch to the Fortran 95 formats for data files whenever possible.

By default, the **f90**, **xlF90**, **xlF90_r**, and **xlF90_r7** commands do not conform completely to the Fortran 90 standard. Also, by default, the **f95**, **xlF95**, **xlF95_r**, and **xlF95_r7** commands do not conform completely to the Fortran 95 standard. If you need full compliance, compile with any of the following additional compiler options (and suboptions):

```
-qnodirective -qnoescape -qextname -qfloat=nomaf:rndsngl:nofold -qnoswapomp  
-qlanglvl=90std -qlanglvl=95std
```

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

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

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

Compiling XL Fortran SMP programs

You can use the **xlF_r**, **xlF_r7**, **xlF90_r**, **xlF90_r7**, **xlF95_r**, or **xlF95_r7** command to compile XL Fortran SMP programs. The **xlF_r** and **xlF_r7** commands are similar to the **xlF** command; the **xlF90_r** and **xlF90_r7** commands are similar to the **xlF90** command; the **xlF95_r** and **xlF95_r7** commands are similar to the **xlF95** command. The main difference is that the thread-safe components (libraries, **crt0_r.o**, and so on) are used to link and bind the object files if you specify the **xlF_r**, **xlF_r7**, **xlF90_r**, **xlF90_r7**, **xlF95_r**, or **xlF95_r7** command.

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

Levels of POSIX pthreads API support

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

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

- To compile and then link your program with the 1003.1-1996 standard interface libraries, use the `xlfr`, `xlfr90`, or `xlfr95` command. For example, you could specify:

```
xlfr95_r test.f
```

- To compile and then link your program with the Draft 7 interface libraries, use the `xlfr7`, `xlfr90_r7`, or `xlfr95_r7` command. For example, you could specify:

```
xlfr95_r7 test.f
```

Apart from the level of thread support, the `xlfr7`, `xlfr90_r7`, and `xlfr95_r7` commands and the corresponding stanzas in the `xlfr.cfg` configuration file provide the same support as the `xlfr`, `xlfr90_r`, and `xlfr95_r` commands and the corresponding stanzas.

Compilation order for Fortran programs

If you have a program unit, subprogram, or interface body that uses a module, you must first compile the module. If the module and the code that uses the module are in separate files, you must first compile the file that contains the module. If they are in the same file, the module must come before the code that uses it in the file. If you change any entity in a module, you must recompile any files that use that module.

Canceling a compilation

To stop the compiler before it finishes compiling, press **Ctrl+C** in interactive mode, or use the `kill` command.

XL Fortran input files

The input files to the compiler are:

Source Files (.f or .F suffix)

All `.f`, `.f77`, `.f90`, `.f95` and `.F` files are source files for compilation. The compiler compiles source files in the order you specify on the command line. If it cannot find a specified source file, the compiler produces an error message and proceeds to the next file, if one exists. Files with a suffix of `.F` are passed through the C preprocessor (`cpp`) before being compiled.

Include files also contain source and often have different suffixes from `.f`.

Related information: See “Passing Fortran files through the C preprocessor” on page 29.

The `fsuffix` and `cppsuffix` attributes in “Customizing the configuration file” on page 11 and the “-qsuffix option” on page 233 let you select a different suffix.

Object Files (.o suffix)

All `.o` files are object files. After the compiler compiles the source files, it uses the `ld` command to link-edit the resulting `.o` files, any `.o` files that you specify as input files, and some of the `.o` and `.a` files in the product and system library directories. It then produces a single executable output file.

Related information: See “Options that control linking” on page 72 and “Linking XL Fortran programs” on page 31.

The `osuffix` attribute, which is described in “Customizing the configuration file” on page 11 and the “-qsuffix option” on page 233, lets you select a different suffix.

Assembler Source Files (.s suffix)

The compiler sends any specified .s files to the assembler (as). The assembler output consists of object files that are sent to the linker at link time.

Related information: The **ssuffix** attribute, which is described in “Customizing the configuration file” on page 11 and the “-qsuffix option” on page 233, lets you select a different suffix.

Archive or Library Files (.a suffix)

The compiler sends any specified library files (.a files) to the linker at link time. There are also AIX and XL Fortran library files in the `/usr/lib` directory that are linked in automatically.

Related information: See “-l option” on page 98, “-L option” on page 97, and “LIBPATH:Setting library search paths” on page 10.

Shared Object Files (.so suffix)

These are object files that can be loaded and shared by multiple processes at run time. When a shared object is specified during linking, information about the object is recorded in the output file, but no code from the shared object is actually included in the output file.

Related information: See “-brtl option” on page 86 and “-bdynamic, -bshared, and -bstatic Options” on page 81.

Configuration Files (.cfg suffix)

The contents of the configuration file determine many aspects of the compilation process, most commonly the default options for the compiler. You can use it to centralize different sets of default compiler options or to keep multiple levels of the XL Fortran compiler present on a system.

The default configuration file is `/etc/xlf.cfg`.

Related information: See “Customizing the configuration file” on page 11 and “-F option” on page 93 for information about selecting the configuration file.

Module Symbol Files: *modulename.mod*

A module symbol file is an output file from compiling a module and is an input file for subsequent compilations of files that **USE** that module. One .mod file is produced for each module, so compiling a single source file may produce multiple .mod files.

Related information: See “-I option” on page 95, “-qmoddir option” on page 191, and “Displaying information inside binary files (what)” on page 302.

Profile Data Files

The **-qpdf1** option produces run-time profile information for use in subsequent compilations. This information is stored in one or more hidden files with names that match the pattern “*.pdf*”.

Related information: See “-qpdf option” on page 198.

XL Fortran Output files

The output files that XL Fortran produces are:

Executable Files: *a.out*

By default, XL Fortran produces an executable file that is named **a.out** in the current directory.

Related information: See “-o option” on page 102 for information on selecting a different name and “-c option” on page 90 for information on generating only an object file.

Object Files: *filename.o*

If you specify the **-c** compiler option, instead of producing an executable file, the compiler produces an object file for each specified **.f** source file, and the assembler produces an object file for each specified **.s** source file. By default, the object files have the same file name prefixes as the source files and appear in the current directory.

Related information: See “-c option” on page 90 and “Linking XL Fortran programs” on page 31. For information on renaming the object file, see “-o option” on page 102.

Assembler Source Files: *filename.s*

If you specify the **-S** compiler option, instead of producing an executable file, the XL Fortran compiler produces an equivalent assembler source file for each specified **.f** source file. By default, the assembler source files have the same file name prefixes as the source files and appear in the current directory.

Related information: See “-S option” on page 260 and “Linking XL Fortran programs” on page 31. For information on renaming the assembler source file, see “-o option” on page 102.

Compiler Listing Files: *filename.lst*

By default, no listing is produced unless you specify one or more listing-related compiler options. The listing file is placed in the current directory, with the same file name prefix as the source file and an extension of **.lst**.

Related information: See “Options that control listings and messages” on page 62.

Module Symbol Files: *modulename.mod*

Each module has an associated symbol file that holds information needed by program units, subprograms, and interface bodies that **USE** that module. By default, these symbol files must exist in the current directory.

Related information: For information on putting **.mod** files in a different directory, see “-qmoddir option” on page 191.

cpp-Preprocessed Source Files: *Ffilename.f*

If you specify the **-d** option when compiling a file with a **.F** suffix, the intermediate file created by the C preprocessor (cpp) is saved rather than deleted.

Related information: See “Passing Fortran files through the C preprocessor” on page 29 and “-d option” on page 92.

Profile Data Files (*.pdf*)

These are the files that the `-qpdf1` option produces. They are used in subsequent compilations to tune optimizations that are based on actual execution results.

Related information: See “-qpdf option” on page 198.

Scope and precedence of option settings

You can specify compiler options in any of three locations. Their scope and precedence are defined by the location you use. (XL Fortran also has comment directives, such as `SOURCEFORM`, that can specify option settings. There is no general rule about the scope and precedence of such directives.)

Location	Scope	Precedence
In a stanza of the configuration file.	All compilation units in all files compiled with that stanza in effect.	Lowest
On the command line.	All compilation units in all files compiled with that command.	Medium
In an <code>@PROCESS</code> directive. (XL Fortran also has comment directives, such as <code>SOURCEFORM</code> , that can specify option settings. There is no general rule about the scope and precedence of such directives.)	The following compilation unit.	Highest

If you specify an option more than once with different settings, the last setting generally takes effect. Any exceptions are noted in the individual descriptions in the “Detailed descriptions of the XL Fortran compiler options” on page 76 and are indexed under “conflicting options”.

Specifying options on the command line

XL Fortran supports the traditional UNIX[®] method of specifying command-line options, with one or more letters (known as flags) following a minus sign:

```
xlf95 -c file.f
```

You can often concatenate multiple flags or specify them individually:

```
xlf95 -cv file.f    # These forms  
xlf95 -c -v file.f # are equivalent
```

(There are some exceptions, such as `-pg`, which is a single option and not the same as `-p -g`.)

Some of the flags require additional argument strings. Again, XL Fortran is flexible in interpreting them; you can concatenate multiple flags as long as the flag with an argument appears at the end. The following example shows how you can specify flags:

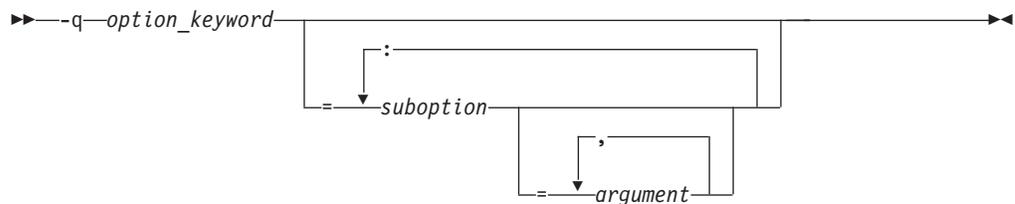
```

# All of these commands are equivalent.
xlf95 -g -v -o montecarlo -p montecarlo.f
xlf95 montecarlo.f -g -v -o montecarlo -p
xlf95 -g -v montecarlo.f -o montecarlo -p
xlf95 -g -v -omontecarlo -p montecarlo.f
# Because -o takes a blank-delimited argument,
# the -p cannot be concatenated.
xlf95 -gvomontecarlo -p montecarlo.f
# Unless we switch the order.
xlf95 -gvpomontecarlo montecarlo.f

```

If you are familiar with other compilers, particularly those in the XL family of compilers, you may already be familiar with many of these flags.

You can also specify many command-line options in a form that is intended to be easy to remember and make compilation scripts and makefiles relatively self-explanatory:



This format is more restrictive about the placement of blanks; you must separate individual `-q` options by blanks, and there must be no blank between a `-q` option and a following argument string. Unlike the names of flag options, `-q` option names are not case-sensitive except that the `q` must be lowercase. Use an equal sign to separate a `-q` option from any arguments it requires, and use colons to separate suboptions within the argument string.

For example:

```
xlf95 -qddim -qXREF=full -qfloat=nomaf:rsqrt -03 -qcache=type=c:level=1 file.f
```

Specifying options in the source file

By putting the `@PROCESS` compiler directive in the source file, you can specify compiler options to affect an individual compilation unit. The `@PROCESS` compiler directive can override options specified in the configuration file, in the default settings, or on the command line.



`option` is the name of a compiler option without the `-q`.

`suboption`

is a suboption of a compiler option.

In fixed source form, `@PROCESS` can start in column 1 or after column 6. In free source form, the `@PROCESS` compiler directive can start in any column.

You cannot place a statement label or inline comment on the same line as an `@PROCESS` compiler directive.

By default, option settings you designate with the `@PROCESS` compiler directive are effective only for the compilation unit in which the statement appears. If the file has more than one compilation unit, the option setting is reset to its original state before the next unit is compiled. Trigger constants specified by the `DIRECTIVE` option are in effect until the end of the file (or until `NODIRECTIVE` is processed).

The `@PROCESS` compiler directive must usually appear before the first statement of a compilation unit. The only exceptions are when specifying `SOURCE` and `NOSOURCE`; you can put them in `@PROCESS` directives anywhere in the compilation unit.

Passing command-line options to the "ld" or "as" commands

Because the compiler automatically executes other commands, such as `ld` and `as`, as needed during compilation, you usually do not need to concern yourself with the options of those commands. If you want to choose options for these individual commands, you can do one of the following:

- Include linker options on the compiler command line. When the compiler does not recognize a command-line option other than a `-q` option, it passes the option on to the linker:

```
xlF95 -berok file.f # -berok is passed to ld
```

- Use the `-W` compiler option to construct an argument list for the command:

```
xlF95 -Wl,-berok file.f # -berok is passed to ld
```

In this example, the `ld` option `-berok` is passed to the linker (which is denoted by the `l` in the `-Wl` option) when the linker is executed.

This form is more general than the previous one because it works for the `as` command and any other commands called during compilation, by using different letters after the `-W` option.

- Edit the configuration file `/etc/xlf.cfg`, or construct your own configuration file. You can customize particular stanzas to allow specific command-line options to be passed through to the assembler or linker.

For example, if you include these lines in the `xlF95` stanza of `/etc/xlf.cfg`:

```
asopt = "w"  
ldopt = "m"
```

and issue this command:

```
xlF95 -wm -Wa,-x -Wl,-s produces_warnings.s uses_many_symbols.f
```

the file `produces_warnings.s` is assembled with the options `-w` and `-x` (issue warnings and produce cross-reference), and the object files are linked with the options `-m` and `-s` (write list of object files and strip final executable file).

Related information: See “`-W` option” on page 266 and “Customizing the configuration file” on page 11.

Compiling for specific architectures

You can use `-qarch` and `-qtune` to target your program to instruct the compiler to generate code specific to a particular architecture. This allows the compiler to take advantage of machine-specific instructions that can improve performance. The `-qarch` option determines the architectures on which the resulting programs can run. The `-qtune` and `-qcache` options refine the degree of platform-specific optimization performed.

By default, the **-qarch** setting produces code using only instructions common to all supported architectures, with resultant settings of **-qtune** and **-qcache** that are relatively general. To tune performance for a particular processor set or architecture, you may need to specify different settings for one or more of these options. The natural progression to try is to use **-qarch**, and then add **-qtune**, and then add **-qcache**. Because the defaults for **-qarch** also affect the defaults for **-qtune** and **-qcache**, the **-qarch** option is often all that is needed.

If the compiling machine is also the target architecture, **-qarch=auto** will automatically detect the setting for the compiling machine. For more information on this compiler option setting, see “-qarch option” on page 114 and also **-O4** and **-O5** under the “-O option” on page 100.

If your programs are intended for execution mostly on particular architectures, you may want to add one or more of these options to the configuration file so that they become the default for all compilations.

Passing Fortran files through the C preprocessor

A common programming practice is to pass files through the C preprocessor (**cpp**). **cpp** can include or omit lines from the output file based on user-specified conditions (“conditional compilation”). It can also perform string substitution (“macro expansion”).

XL Fortran can use **cpp** to preprocess a file before compiling it. If you are also using one of the optimizing preprocessors, **cpp** is called before any of the other preprocessors.

To call **cpp** for a particular file, use a file suffix of **.F**. If you specify the **-d** option, each **.F** file *filename.F* is preprocessed into an intermediate file *Ffilename.f*, which is then compiled. If you do not specify the **-d** option, the intermediate file name is */tmpdir/F8xxxxxx*, where *x* is an alphanumeric character and *tmpdir* is the contents of the **TMPDIR** environment variable or, if you have not specified a value for **TMPDIR**, **/tmp**. You can save the intermediate file by specifying the **-d** compiler option; otherwise, the file is deleted. If you only want to preprocess and do not want to produce object or executable files, specify the **-qnoobject** option also.

When XL Fortran uses **cpp** for a file, the preprocessor will emit **#line** directives unless you also specify the **-d** option. The **#line** directive associates code that is created by **cpp** or any other Fortran source code generator with input code that you create. The preprocessor may cause lines of code to be inserted or deleted. Therefore, the **#line** directives that it emits can be useful in error reporting and debugging, because they identify the source statements found in the preprocessed code by listing the line numbers that were used in the original source.

The **_OPENMP** C preprocessor macro can be used to conditionally include code. This macro is defined when the C preprocessor is invoked and when you specify the **-qsmp=omp** compiler option. An example of using this macro follows:

```

program par_mat_mul
  implicit none
  integer(kind=8)                :: i,j,nthreads
  integer(kind=8),parameter      :: N=60
  integer(kind=8),dimension(N,N) :: Ai,Bi,Ci
  integer(kind=8)                :: Sumi
#ifdef _OPENMP
  integer omp_get_num_threads
#endif

```

```

        common/data/ Ai,Bi,Ci
!$OMP threadprivate (/data/)

!$omp parallel
    forall(i=1:N,j=1:N) Ai(i,j) = (i-N/2)**2+(j+N/2)
    forall(i=1:N,j=1:N) Bi(i,j) = 3-((i/2)+(j-N/2)**2)
!$omp master
#ifdef _OPENMP
    nthreads=omp_get_num_threads()
#else
    nthreads=8
#endif
!$omp end master
!$omp end parallel

!$OMP parallel default(private),copyin(Ai,Bi),shared(nthreads)
!$omp do
    do i=1,nthreads
        call imat_mul(Sumi)
    enddo
!$omp end do
!$omp end parallel

end

```

See *Conditional Compilation* in the *Language Elements* section of the *XL Fortran Language Reference* for more information on conditional compilation.

To customize **cpp** preprocessing, the configuration file accepts the attributes **cpp**, **cppsuffix**, and **cppoptions**.

The letter F denotes the C preprocessor with the **-t** and **-W** options.

Related information: See “-d option” on page 92, “-t option” on page 261, “-W option” on page 266, and “Customizing the configuration file” on page 11.

cpp Directives for XL Fortran programs

Macro expansion can have unexpected consequences that are difficult to debug, such as modifying a **FORMAT** statement or making a line longer than 72 characters in fixed source form. Therefore, we recommend using **cpp** primarily for conditional compilation of Fortran programs. The **cpp** directives that are most often used for conditional compilation are **#if**, **#ifdef**, **#ifndef**, **#elif**, **#else**, and **#endif**.

Passing options to the C preprocessor

Because the compiler does not recognize **cpp** options other than **-I** directly on the command line, you must pass them through the **-W** option. For example, if a program contains **#ifdef** directives that test the existence of a symbol named **AIXV4**, you can define that symbol to **cpp** by compiling with a command like:

```
xlf95 conditional.F -WF,-DAIXV4
```

Avoiding preprocessing problems

Because Fortran and C differ in their treatment of some sequences of characters, be careful when using **/*** or ***/**. These might be interpreted as C comment delimiters, possibly causing problems even if they occur inside Fortran comments. Also be careful when using three-character sequences that begin with **??** (which might be interpreted as C trigraphs).

Consider the following example:

```
program testcase
character a
character*4 word
a = '?'
word(1:2) = '??'
print *, word(1:2)
end program testcase
```

If the preprocessor matches your character combination with the corresponding trigraph sequence, your output may not be what you expected.

If your code does *not* require the use of the XL Fortran compiler option **-qnoescape**, a possible solution is to replace the character string with an escape sequence **word(1:2) = '\?\?'**. However, if you are using the **-qnoescape** compiler option, this solution will not work. In this case, you require a **cpp** that will ignore the trigraph sequence. XL Fortran uses the **cpp** that is found in **/usr/ccs/lib/cpp**. This is the standard **cpp**. It is **ISO C** compliant and therefore recognizes trigraph sequences.

On the AIX operating system, **cpp** has the option **-qlanglvl=classic**. Therefore, compile the trigraph example by using the following command:

```
xlf95 tst.F -d -v -WF,-qlanglvl=classic
```

This invokes **cpp tst.F -qlanglvl=classic**.

Linking XL Fortran programs

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 object files **file1.o** and **file3.o**, then all object files are submitted to the linker to produce one executable.

After linking, follow the instructions in “Running XL Fortran programs” on page 37 to execute the program.

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 (file1.o, file3.o)
xlf95 file1.o file2.o file3.o # Link object files with appropriate libraries
```

It is often best to execute the linker through the compiler invocation command, because it passes some extra **ld** options and library names to the linker automatically.

Linking 32-bit SMP object files using the ld command

To use the **ld** command to link an SMP program, follow these guidelines:

- Do not use the **-e** flag to change the default starting point of the executable output file (**__start**). If you use other starting points, your results will be unpredictable.
- Specify the following options and files with the **ld** command:
 - **-bh:4, -bpT:0x10000000, -bpD:0x20000000**.

- **-lxf** before any other libraries or files on the command line if you are linking any object files compiled by XL Fortran Version 2.
- The object file that contains the system startup routine:
 - **crt0_r.o** for a program that was not profiled.
 - **mcrt0_r.o** for a program that was profiled with the **-p** option.
 - **gcr0_r.o** for a program that was profiled with the **-pg** option.
 - Link with the startup files in **/usr/lib**.
- Compiler and system libraries, in the following order:
 - **-lxfpthreads_compat** (for POSIX pthreads Draft 7 support), **-lxf90**, **-lxf**, **-lxlsmp**, **-lc**, and either:
 - **-lpthreads** (for POSIX pthreads 1003.1-1996 standard support).
 - **-lpthreads_compat** followed by **-lpthreads** (for POSIX pthreads Draft 7 support).
 - You only need to specify **-lxlsmp** if you are compiling with **-qsmp**.
 - If you use the **-qautodbl** option, specify some extra libraries that are listed in “-qautodbl option” on page 121.
 - If you use the **-qpdf1** compiler option, specify **-lxlopt**.
 - If you use the **-qhot=vector** suboption, specify **-lxlopt**.

On AIX Version 5.1 and higher, the default POSIX pthreads API is the 1003.1-1996 standard. If you had a program called **mytest** and you wanted to obtain access to the functions in the 1003.1-1996 standard POSIX pthreads API on AIX Version 5.1, you would link with the **libpthreads.a** library, using something similar to the following command:

```
ld -bh:4 -bpT:0x10000000 -bpD:0x20000000 /lib/crt0_r.o mytest.o -lxf90
-lxf -lxlsmp -lm -lc -lpthreads -o mytest
```

The 1003.1-1996 standard is not fully compatible with Draft 7. If you have programs that require the Draft 7 interface, link your programs with the **libpthreads_compat.a** and **liblxfpthreads_compat.a** libraries (which provide compatibility support) followed by the **libpthreads.a** library. For example, if you have a program called **mytest** that was written to use the Draft 7 interface, on AIX Version 5.1, you would use something similar to the following command:

```
ld -bh:4 -bpT:0x10000000 -bpD:0x20000000 /lib/crt0_r.o mytest.o
-lxfpthreads_compat -lxf90 -lxf -lxlsmp -lm -lc
-lpthreads_compat -lpthreads -o mytest
```

The configuration file **/etc/xlf.cfg** lists these default libraries and linker options. By doing a sample compilation with the **-#** option, you can see exactly how the compiler would run the linker.

See the *AIX Commands Reference* for a description of the linker options.

Linking 64-bit SMP object files using the **ld** command

To use the **ld** command to link a 64-bit SMP program, follow these guidelines:

- Do not use the **-e** flag to change the default starting point of the executable output file (**__start**). If you use other starting points, your results will be unpredictable.
- Specify the following options and files with the **ld** command:
 - On AIX 5.1, **-bh:4**, **-bpT:0x10000000**, **-bpD:0x20000000**, **-b64**
 - On AIX 5.2 and above, **-bh:4**, **-bpT:0x100000000**, **-bpD:0x110000000**, **-b64**
 - The object file that contains the system startup routine:
 - **crt0_64.o** for a program that was not profiled.
 - **mcrt0_64.o** for a program that was profiled with the **-p** option.
 - **gcr0_64.o** for a program that was profiled with the **-pg** option.

- Link with the startup files in **/usr/lib**.
- Compiler and system libraries:
 - **-lxf90**, **-lxlsm**, **-lm**, **-lc**, and **-lpthreads**, in that order (you only need **-lxlsm** if you compile with the **-qsmp** option).
 - If you use the **-qautodbl** option, specify one of the libraries that are listed in the suboptions of “-qautodbl option” on page 121.
 - If you use the **-qpdf1** compiler option, specify **-lxlopt**.
 - If you use the **-qhot=vector** suboption, specify **-lxlopt**.

For example, to link the object files `smpfile1.o` and `smpfile2.o` on AIX 5.1, you could specify the following:

```
ld -bh:4 -bpT:0x10000000 -bpD:0x20000000 -b64 /lib/crt0_64.o -lxf90
-lxlsm -lm -lc smpfile1.o smpfile2.o
```

The configuration file `/etc/xlf.cfg` lists these default libraries and linker options. By doing a sample compilation with the `-#` option, you can see exactly how the compiler would run the linker.

See the *AIX Commands Reference* for a description of the linker options.

Linking 32-bit non-SMP object files using the `ld` command

To use the `ld` command to link non-SMP object files in a 32-bit environment, follow these guidelines:

- Do not use the **-e** flag to change the default starting point of the executable output file (`__start`). If you use other starting points, your results will be unpredictable.
- Specify the following options and files with the `ld` command:
 - **-bh:4**, **-bpT:0x10000000**, **-bpD:0x20000000**.
 - **-lxf** before any other libraries or files on the command line if any object files compiled by XL Fortran Version 2 are being linked.
 - The object file that contains the system startup routine:
 - **crt0.o** for a program that was not profiled.
 - **mcrt0.o** for a program that was profiled with the **-p** option.
 - **gcr0.o** for a program that was profiled with the **-pg** option.
 - Link with the startup files in **/usr/lib**.
- Compiler and system libraries:
 - **-lxf90**, **-lm**, and **-lc** in that order.
 - If you use the **-qautodbl** option, specify some extra libraries that are listed in “-qautodbl option” on page 121.
 - If you use the **-qpdf1** compiler option, specify **-lxlopt**.
 - If you use the **-qhot=vector** suboption, specify **-lxlopt**.

For example, to link the object files `file1.o` and `file2.o`, you could specify the following:

```
ld -bh:4 -bpT:0x10000000 -bpD:0x20000000 /lib/crt0.o -lxf90 -lm -lc
file1.o file2.o
```

The configuration file `/etc/xlf.cfg` lists these default libraries and linker options. By doing a sample compilation with the `-#` option, you can see exactly how the compiler would run the linker.

See the *AIX Commands Reference* for a description of the linker options.

Linking 64-bit non-SMP object files using the ld command

To use the **ld** command to link non-SMP object files in a 64-bit environment, follow these guidelines:

- Do not use the **-e** flag to change the default starting point of the executable output file (`__start`). If you use other starting points, your results will be unpredictable.
- Specify the following options and files with the **ld** command:
 - On AIX 5.1, **-bh:4**, **-bpT:0x10000000**, **-bpD:0x20000000**, **-b64**.
 - On AIX 5.2 and above, **-bh:4**, **-bpT:0x100000000**, **-bpD:0x110000000**, **-b64**
 - The object file that contains the system startup routine:
 - **crt0_64.o** for a program that was not profiled.
 - **mcrto_64.o** for a program that was profiled with the **-p** option.
 - **gcrt0_64.o** for a program that was profiled with the **-pg** option.
 - Link with the startup files in **/usr/lib**.
 - Compiler and system libraries:
 - **-lxf90**, **-lm**, and **-lc** in that order.
 - If you use the **-qautodbl** option, specify some extra libraries that are listed in “-qautodbl option” on page 121.
 - If you use the **-qpdf1** compiler option, specify **-lxlopt**.
 - If you use the **-qhot=vector** suboption, specify **-lxlopt**.

For example, to link the object files `file1.o` and `file2.o` on AIX 5.1, you could specify the following:

```
ld -bh:4 -bpT:0x10000000 -bpD:0x20000000 -b64 /lib/crt0_64.o -lxf90 -lm
-lc file1.o file2.o
```

The configuration file `/etc/xf.cfg` lists these default libraries and linker options. By doing a sample compilation with the **-#** option, you can see exactly how the compiler would run the linker.

See the *AIX Commands Reference* for a description of the linker options.

Passing options to the ld command

If you need to link with **ld** options that are not part of the XL Fortran default, you can include those options on the compiler command line:

```
xlf95 -bhalt:2 -K -r file.f # xlf95 passes all these options to ld
```

The compiler passes unrecognized options, except **-q** options, to the **ld** command.

Checking for interface errors at link time

If you specify the **-qextchk** compiler option, the linker may refuse to link object files containing mismatching procedure interfaces or common block definitions, allowing you to find these errors at link time, instead of trying to debug incorrect results.

If the linking problem can be isolated to a few names that do not resolve, perhaps because of uppercase letters in C names or trailing underscores added by the **-qextname** option, you can use the **-brename** linker option to change just those names:

```
xlf95 -brename:Old_Link_Name,new_link_name fort_prog.o c_prog.o
```

Related information: See “-qextchk option” on page 145, “-U option” on page 262, and “-qextname option” on page 147.

Linking new objects with existing ones

If you have `.o` or other object files that you compiled with an earlier version of XL Fortran, you can link them with object files that you compile with XL Fortran Version 10, subject to the following notes. The main XL Fortran library is `libxlf90.a`, but calls to older entry points in `libxlf.a` are still possible; the calls are passed to the new entry points in the main libraries, which makes the resulting programs slower than if everything is recompiled.

Notes:

1. You must explicitly specify the XL Fortran `libxlf.a` library as part of the link step, preferably by including the option `-lxf`.
2. For safety, always put `-lxf` as the first option after the compiler command so that the library is linked before any user object files. Doing so ensures that the new I/O routines override any existing ones in statically linked object files.
3. When you relink old object files, the I/O routines in the resulting program differ in some ways from the behavior of XL Fortran Version 2. To make the resulting program work as you expect, you may need to change some of the run-time settings in “Setting run-time options” on page 40 (particularly the `namelist` setting) or to recompile the source files with the “`-qxlf77` option” on page 252. Some changed I/O details cannot be switched to the old behavior at all.
4. You cannot link files that you compiled with the XL Fortran Version 4 level of IPA with files that you compiled with the XL Fortran Version 6 level or later of IPA.
5. You cannot link 64-bit objects compiled with XL Fortran version 7.1.0.1, or lower. The object format has changed on AIX Version 5.1.
6. You cannot link `pdf` files that you created with `-qpdf1` and Version 5.1.0 or earlier levels of XL Fortran with `pdf` files that you created with `-qpdf1` and XL Fortran Version 7.1 or higher. However, you can link object files that you created with `-qpdf2` and XL Fortran Version 7.1 or higher with object files that you created with `-qpdf2` and earlier levels of XL Fortran.

Relinking an existing executable file

Because the linker accepts executable files as input, 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: If this type of linking is done incorrectly, it can result in interface errors and other problems. Therefore, you should not try it unless you are experienced 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 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 (as explained in “Avoiding naming conflicts during linking”). 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.

You can use **-b** linker options on the compiler command line to create statically linked object files:

```
xlf95 -bnso -bI:/usr/lib/syscalls.exp file1.f file2.f
```

You must also specify **-bI:/usr/lib/threads.exp** when you are statically linking with the **xlf_r**, **xlf_r7**, **xlf90_r**, **xlf90_r7**, **xlf95_r**, or **xlf95_r7** command.

If you are using Asynchronous I/O, you must also specify **-bI:/usr/lib/aio.exp**.

The **-bnso** option places the library procedures that your program references into the program’s object file. Files with a suffix of **.exp** specify the names of system routines that must be imported to your program from the system.

An alternative that requires less disk space is to link any XL Fortran libraries statically but to leave references to other system libraries dynamic. This example statically links just the XL Fortran libraries:

```
# Build a temporary object from the Fortran library:
ld -r -o libtmp.o -bnso -lxlf90
# Build the application with this object on the command line:
xlf95 -o appl appl1.o appl2.o libtmp.o
```

Avoiding naming conflicts during linking

If you define an external subroutine, external function, or common block with the same name as a run-time subprogram, your definition of that name may be used in its place, or it may cause a link-edit error.

Try the following general solution to help avoid these kinds of naming conflicts:

- Compile all files with the **-qextname** option. It adds an underscore to the end of the name of each global entity, making it distinct from any names in the system libraries.

Note: When you use this option, you do not need to use the final underscore in the names of Service and Utility Subprograms, such as **dtime_** and **flush_**.

- Link your programs dynamically, which is the default. Many naming conflicts only apply to statically linked programs.
- Order the names of libraries and object files on the command line so that the one that should take precedence comes first. For example, to make names in **libxlf90.a** take precedence over duplicate names in an object file, specify **-lxlf90** first on the command line.

If you do not use the **-qextname** option, you must take the following extra precautions to avoid conflicts with the names of the external symbols in the XL Fortran and system libraries:

- Do not name a subroutine or function **main**, because XL Fortran defines an entry point **main** to start your program.
- Do not use *any* global names that begin with an underscore. In particular, the XL Fortran libraries reserve all names that begin with **_xl**.
- Do not use names that are the same as names in the XL Fortran library or one of the system libraries. To determine which names are not safe to use in your program, run the **nm** command on any libraries that are linked into the program and search the output for names you suspect might also be in your program.
- If your program calls certain XLF-provided routines, some restrictions apply to the common block and subprogram names that you can use:

XLF-Provided Function Name	Common Block or Subprogram Name You Cannot Use
mclock	times
rand	irand

Be careful not to use the names of subroutines or functions without defining the actual routines in your program. If the name conflicts with a name from one of the libraries, the program could use the wrong version of the routine and not produce any compile-time or link-time errors.

If different versions of a routine occur in more than one library or object file, be careful to use the specific version that you want. Specify the file with the correct version as the first file on the command line or in the configuration file. If the file is a library, specify the appropriate **-l** option first on the command line. This technique does not apply to references between routines that are in the same shared library or to routines that are explicitly imported from one shared library to another.

Running XL Fortran programs

The default file name for the executable program is **a.out**. You can select a different name with the **-o** compiler option. You should avoid giving your programs the same names as system or shell commands (such as **test** or **cp**), as you could accidentally execute the wrong command. If a name conflict does occur, you can execute the program by specifying a path name, such as **./test**.

You can run a program by entering the path name and file name of an executable file along 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.

Running previously compiled programs

Statically linked programs that you compiled with levels of XL Fortran prior to Version 10.1 should continue to run with no change in performance or behavior. They may not run on a system with a level of the operating system different from the system on which they were compiled.

If you have dynamically linked programs compiled by XL Fortran Versions 2 through 9, you can run them on systems with the XL Fortran Version 10 libraries. The programs will use the current compiler data formats and I/O behavior, which are somewhat different from those of XL Fortran Version 2.

Compiling and executing on different systems

If you want to move an XL Fortran executable file to a different system for execution, you can link statically and copy the program, and optionally the run-time message catalogs. Alternatively, you can link dynamically and copy the program as well as the XL Fortran libraries if needed and optionally the run-time message catalogs. For non-SMP programs, **libxlf90.a** is usually the only XL Fortran library needed. For SMP programs, you will usually need at least the **libxlf90.a** and **libxlsmp.a** libraries. **libxlf.a** is only needed if the program has any XL Fortran Version 1 or 2 object files linked in. **libxlfpm*.a** and **libxlfpad.a** are only needed if the program is compiled with the **-qautodbl** option. If your application has dependencies on **libhmd.a**, refer to “Using debug memory routines for XL Fortran” on page 287 for more details on library dependencies.

For a dynamically linked program to work correctly, the XL Fortran libraries and the operating system on the execution system must be at either the same level or a more recent level than on the compilation system.

For a statically linked program to work properly, the operating-system level may need to be the same on the execution system as it is on the compilation system.

Related information: See “Dynamic and Static Linking” on page 36.

POSIX Pthreads binary compatibility

The XL Fortran compiler and run-time library provide binary compatibility in the following areas:

- Executable file binary compatibility. If you created an executable file that had dependencies on the pthreads Draft 7 API (for example, you used XL Fortran Version 5.1.0 or AIX Version 4.2.1), you can upgrade your system to use XL Fortran Version 10.1.0 or AIX Version 5.1 and run your executable file without first recompiling and relinking your program.
- Object file or archive library binary compatibility. If you created an object file or archive library that had dependencies on the Draft 7 pthreads API, you can continue to use that object file or archive library with the Draft 7 interface if you move from AIX Version 4.2.1 to AIX Version 5.1. For example, if you have a source file called **test.f** that uses a shared or static archive library called

libmy_utility.a (which was created with the Draft 7 interface), you would enter something similar to the following command on AIX Version 5.1:

```
xlf95_r7 test.f -lmy_utility -o a.out
```

You do not need to regenerate **libmy_utility.a** before using it on AIX Version 5.1.

There are, however, restrictions on binary compatibility. XL Fortran supports combinations of Draft 7 and 1003.1-1996 standard object files in some instances. For example, if you used XL Fortran Version 5.1.0 to create a library, that library uses the Draft 7 pthreads API. An application that you build with that library can use either the Draft 7 pthreads API or the 1003.1-1996 standard pthreads API as long as the portions of the complete application built with the Draft 7 pthreads API do not share any pthreads data objects (such as mutexes or condition variables) with the portions built with the 1003.1-1996 standard pthreads API. If any such objects need to be used across portions of an application that are compiled with different levels of the pthreads API, the final application needs to use either the Draft 7 pthreads API or the 1003.1-1996 standard pthreads API across the entire application. You can do this in one of two ways:

- Build the application by using the **xlf_r7**, **xlf90_r7**, or **xlf95_r7** command, so that it uses the Draft 7 pthreads API.
- Build both the library and the rest of the application by using the **xlf_r**, **xlf90_r**, or **xlf95_r** command.

Run-time libraries for POSIX pthreads support

There are three run-time libraries that are connected with POSIX thread support. The **libxlf90.a** library is the Fortran run-time library that supports both non-threaded and threaded applications. The **libxlsmp.a** library is the SMP run-time library.

The following libraries are used:

/lib/libxlf90.a	Provides 1003.1-1996 standard 32-bit and 64-bit support.
/lib/libxlsmp.a	Provides 1003.1-1996 standard 32-bit and 64-bit support.
/lib/libxlfpthrds_compat.a	Provides Draft 7 32-bit support.

XL Fortran supplies the following directories for **.mod** files:

/usr/lpp/xlf/include_d7	Provides Draft 7 32-bit support.
/usr/lpp/xlf/include	Provides 1003.1-1996 standard 32-bit and 64-bit support.

Depending on the invocation command, and in some cases, the compiler option, the appropriate set of libraries and include files for thread support is bound in. For example:

The environment variable is checked when the program first encounters one of the following conditions:

- An I/O statement is executed.
- The **RANDOM_SEED** procedure is executed.
- An **ALLOCATE** statement needs to issue a run-time error message.
- A **DEALLOCATE** statement needs to issue a run-time error message.
- The multi-threaded implementation of the **MATMUL** procedure is executed.

Changing the **XLFRTEOPTS** environment variable during the execution of a program has no effect on the program.

The **SETRTEOPTS** procedure (which is defined in the *XL Fortran Language Reference*) accepts a single-string argument that contains the same name-value pairs as the **XLFRTEOPTS** environment variable. It overrides the environment variable and can be used to change settings during the execution of a program. The new settings remain in effect for the rest of the program unless changed by another call to **SETRTEOPTS**. Only the settings that you specified in the procedure call are changed.

You can specify the following run-time options with the **XLFRTEOPTS** environment variable or the **SETRTEOPTS** procedure:

buffering={**enable** | **disable_preconn** | **disable_all**}

Determines whether the XL Fortran run-time library performs buffering for I/O operations.

The library reads data from, or writes data to the file system in chunks for **READ** or **WRITE** statements, instead of piece by piece. The major benefit of buffering is performance improvement.

If you have applications in which Fortran routines work with routines in other languages or in which a Fortran process works with other processes on the same data file, the data written by Fortran routines may not be seen immediately by other parties (and vice versa), because of the buffering. Also, a Fortran **READ** statement may read more data than it needs into the I/O buffer and cause the input operation performed by a routine in other languages or another process that is supposed to read the next data item to fail. In these cases, you can use the **buffering** run-time option to disable the buffering in the XL Fortran run-time library. As a result, a **READ** statement will read in exactly the data it needs from a file and the data written by a **WRITE** statement will be flushed out to the file system at the completion of the statement.

Note: I/O buffering is always enabled for files on sequential access devices (such as pipes, terminals, sockets, and tape drives). The setting of the **buffering** option has no effect on these types of files.

If you disable I/O buffering for a logical unit, you do not need to call the Fortran service routine **flush_** to flush the contents of the I/O buffer for that logical unit.

The suboptions for **buffering** are as follows:

enable

The Fortran run-time library maintains an I/O buffer for each connected logical unit. The current read-write file pointers that the run-time library maintains might not be synchronized with the read-write pointers of the corresponding files in the file system.

disable_preconn

The Fortran run-time library does not maintain

an I/O buffer for each preconnected logical unit (0, 5, and 6). However, it does maintain I/O buffers for all other connected logical units. The current read-write file pointers that the run-time library maintains for the preconnected units are the same as the read-write pointers of the corresponding files in the file system.

disable_all

The Fortran run-time library does not maintain I/O buffers for any logical units. You should not specify the **buffering=disable_all** option with Fortran programs that perform asynchronous I/O.

In the following example, Fortran and C routines read a data file through redirected standard input. First, the main Fortran program reads one integer. Then, the C routine reads one integer. Finally, the main Fortran program reads another integer.

Fortran main program:

```
integer(4) p1,p2,p3
print *, 'Reading p1 in Fortran...'
read(5,*) p1
call c_func(p2)
print *, 'Reading p3 in Fortran...'
read(5,*) p3
print *, 'p1 p2 p3 Read: ', p1,p2,p3
end
```

C subroutine (c_func.c):

```
#include <stdio.h>
void
c_func(int *p2)
{
    int n1 = -1;

    printf("Reading p2 in C...\n");
    setbuf(stdin, NULL); /* Specifies no buffering for stdin */
    fscanf(stdin,"%d",&n1);
    *p2=n1;
    fflush(stdout);
}
```

Input data file (infile):

```
11111
22222
33333
44444
```

The main program runs by using infile as redirected standard input, as follows:

```
$ main < infile
```

If you turn on **buffering=disable_preconn**, the results are as follows:

```
Reading p1 in Fortran...
Reading p2 in C...
Reading p3 in Fortran...
p1 p2 p3 Read: 11111 22222 33333
```

If you turn on **buffering=enable**, the results are unpredictable.

cnvrr={yes | no}

If you set this run-time option to **no**, the program does not obey the **IOSTAT=** and **ERR=** specifiers for I/O statements that encounter conversion errors. Instead, it performs default recovery actions (regardless of the setting of **err_recovery**) and may issue warning messages (depending on the setting of **xrf_messages**).

Related information: For more information about conversion errors, see *Data Transfer Statements* in the *XL Fortran Language Reference*. For more information about **IOSTAT** values, see *Conditions and IOSTAT Values* in the *XL Fortran Language Reference*.

cpu_time_type={usertime | systime | alltime | total_usertime | total_systime | total_alltime}

Determines the measure of time returned by a call to **CPU_TIME(TIME)**.

The suboptions for **cpu_time_type** are as follows:

usertime

Returns the user time of a process. (For a definition of user time, see the *AIX Performance Management Guide*).

systime

Returns the system time of a process. (For a definition of system time, see the *AIX Performance Management Guide*).

alltime

Returns the sum of the user and system time of a process.

total_usertime

Returns the total user time of a process. The total user time is the sum of the user time of a process and the total user times of its child processes, if any.

total_systime

Returns the total system time of a process. The total system time is the sum of the system time of the current process and the total system times of its child processes, if any.

total_alltime

Returns the total user and system time of a process. The total user and system time is the sum of the user and system time of the current process and the total user and system times of their child processes, if any.

default_recl={64 | 32}

Allows you to determine the default record size for sequential files opened without a **RECL=** specifier. The suboptions are as follows:

64 Uses a 64-bit value as the default record size.

32 Uses a 32-bit value as the default record size.

The **default_recl** run-time option applies only in 64-bit mode. In 32-bit mode, **default_recl** is ignored and the record size is 32-bit.

Use **default_recl** when porting 32-bit programs to 64-bit mode where a 64-bit record length will not fit into the specified integer variable. Consider the following:

```
INTEGER(4) I
OPEN (11)
INQUIRE (11, RECL=i)
```

A run-time error occurs in the above code sample in 64-bit mode when **default_recl=64**, since the default record length of $2^{63}-1$ does not fit into the 4-byte integer I. Specifying **default_recl=32** ensures a default record size of $2^{31}-1$, which fits into I.

For more information on the **RECL=** specifier, see the **OPEN** statement in the *XL Fortran Language Reference*.

erroeof={yes | no}

Determines whether the label specified by the **ERR=** specifier is to be branched to if no **END=** specifier is present when an end-of-file condition is encountered.

err_recovery={yes | no}

If you set this run-time option to **no**, the program stops if there is a recoverable error while executing an I/O statement with no **IOSTAT=** or **ERR=** specifiers. By default, the program takes some recovery action and continues when one of these statements encounters a recoverable error. Setting **cnverr** to **yes** and **err_recovery** to **no** can cause conversion errors to halt the program.

iostat_end={extended | 2003std}

Sets the **IOSTAT** values based on the XL Fortran definition or the Fortran 2003 Standard when end-of-file and end-of-record conditions occur. The suboptions are as follows:

extended

Sets the **IOSTAT** variables based on XL Fortran's definition of values and conditions.

2003std

Sets the **IOSTAT** variables based on Fortran 2003's definition of values and conditions.

For example, setting the **iostat_end=2003std** run-time option results in a different **IOSTAT** value from extensions being returned for the end-of-file condition

```
export XLF RTEOPTS=iostat_end=2003std
character(10) ifl
integer(4) aa(3), ios
ifl = "12344321 "
read(ifl, '(3i4)', iostat=ios) aa ! end-of-file condition occurs and
! ios is set to -1 instead of -2.
```

For more information on setting and using **IOSTAT** values, see the **READ**, **WRITE**, and *Conditions and IOSTAT Values* sections in the *XL Fortran Language Reference*.

intrinths={num_threads}

Specifies the number of threads for parallel execution of the **MATMUL** and **RANDOM_NUMBER** intrinsic procedures. The default value for **num_threads** when using the **MATMUL** intrinsic equals the number of processors online. The default value for **num_threads** when using the **RANDOM_NUMBER** intrinsic is equal to the number of processors online*2.

Changing the number of threads available to the **MATMUL** and **RANDOM_NUMBER** intrinsic procedures can influence performance.

langlvl={extended | 90ext | 90std | 95std | 2003std | 2003pure}

Determines the level of support for Fortran standards and extensions to the standards. The values of the suboptions are as follows:

- 90std** Specifies that the compiler should flag any extensions to the Fortran 90 standard I/O statements and formats as errors.
- 90ext** Currently, provides the same level of support as the **extended** suboption. **90ext** was the default suboption prior to XL Fortran Version 7.1. However, this suboption is now obsolete, and to avoid problems in the future, you should start using the **extended** suboption as soon as possible.
- 95std** Specifies that the compiler should flag any extensions to the Fortran 95 standard I/O statements and formats as errors.
- 2003std** Specifies that the compiler should accept all standard I/O statements and formats that the Fortran 95 standard specifies, as well as those Fortran 2003 formats that XL Fortran supports. Anything else is flagged as an error.
- For example, setting the **langlvl=2003std** run-time option results in a run-time error message.
- ```
integer(4) aa(100)
call setrteopts("langlvl=2003std")
... ! Write to a unit without explicitly
... ! connecting the unit to a file.
write(10, *) aa ! The implicit connection to a file does not
... ! conform with Fortran 2003 behavior.
```
- 2003pure** The same as **2003std** except that it also reports errors for any obsolescent Fortran 2003 features used.
- extended** Specifies that the compiler should accept the Fortran 95 language standard, Fortran 2003 features supported by XL Fortran, and extensions, effectively turning off language-level checking.

To obtain support for items that are part of the Fortran 95 standard and are available in XL Fortran as of Version 10.1 (such as namelist comments), you must specify one of the following suboptions:

- **95std**
- **2003std**
- **extended**

The following example contains a Fortran 95 extension (the *file* specifier is missing from the **OPEN** statement):

```
program test1

call setrteopts("langlvl=95std")
open(unit=1,access="sequential",form="formatted")

10 format(I3)

write(1,fmt=10) 123

end
```

Specifying **langlvl=95std** results in a run-time error message.

The following example contains a Fortran 95 feature (namelist comments) that was not part of Fortran 90:

```
program test2

INTEGER I
LOGICAL G
NAMELIST /TODAY/G, I

call setrteopts("langlvl=95std:namelist=new")

open(unit=2,file="today.new",form="formatted", &
 & access="sequential", status="old")

read(2,nml=today)
close(2)

end

today.new:

&TODAY ! This is a comment
I = 123, G=.true. /
```

If you specify **langlvl=95std**, no run-time error message is issued. However, if you specify **langlvl=90std**, a run-time error message is issued.

The **err\_recovery** setting determines whether any resulting errors are treated as recoverable or severe.

**multconn={yes | no}**

Enables you to access the same file through more than one logical unit simultaneously. With this option, you can read more than one location within a file simultaneously without making a copy of the file.

You can only use multiple connections within the same program for files on random-access devices, such as disk drives. In particular, you cannot use multiple connections within the same program for:

- Files have been connected for write-only (**ACTION='WRITE'**)
- Asynchronous I/O
- Files on sequential-access devices (such as pipes, terminals, sockets, and tape drives)

To avoid the possibility of damaging the file, keep the following points in mind:

- The second and subsequent **OPEN** statements for the same file can only be for reading.
- If you initially opened the file for both input and output purposes (**ACTION='READWRITE'**), the unit connected to the file by the first **OPEN** becomes read-only (**ACCESS='READ'**) when the second unit is connected. You must close all of the units that are connected to the file and reopen the first unit to restore write access to it.
- Two files are considered to be the same file if they share the same device and i-node numbers. Thus, linked files are considered to be the same file.

**multconnio={tty | nulldev | combined | no }**

Enables you to connect a device to more than one logical unit. You can then write to, or read from, more than one logical unit that is attached to the same device. The suboptions are as follows:

**combined**

Enables you to connect a combination of null and TTY devices to more than one logical unit.

**nulldev**

Enables you to connect the null device to more than one logical unit.

**tty** Enables you to connect a TTY device to more than one logical unit.

**Note:** Using this option can produce unpredictable results.

In your program, you can now specify multiple **OPEN** statements that contain different values for the **UNIT** parameters but the same value for the **FILE** parameters. For example, if you have a symbolic link called **mytty** that is linked to TTY device **/dev/tty**, you can run the following program when you specify the **multconnio=tty** option:

```
PROGRAM iotest
OPEN(UNIT=3, FILE='mytty', ACTION="WRITE")
OPEN(UNIT=7, FILE='mytty', ACTION="WRITE")
END PROGRAM iotest
```

Fortran preconnects units 0, 5, and 6 to the same TTY device. Normally, you cannot use the **OPEN** statement to explicitly connect additional units to the TTY device that is connected to units 0, 5, and 6. However, this is possible if you specify the **multconnio=tty** option. For example, if units 0, 5, and 6 are preconnected to TTY device **/dev/tty**, you can run the following program if you specify the **multconnio=tty** option:

```
PROGRAM iotest
! /dev/pts/2 is your current tty, as reported by the 'tty' command.
! (This changes every time you login.)
CALL SETRTEOPTS ('multconnio=tty')
OPEN (UNIT=3, FILE='/dev/pts/2')
WRITE (3, *) 'hello' ! Display 'hello' on your screen
END PROGRAM
```

**namelist={new | old}**

Determines whether the program uses the XL Fortran new or old (Version 1) **NAMELIST** format for input and output. The Fortran 90 and Fortran 95 standards require the new format.

**Note:** You may need the **old** setting to read existing data files that contain **NAMELIST** output. However, use the standard-compliant new format in writing any new data files.

With **namelist=old**, the nonstandard **NAMELIST** format is not considered an error by the **langlvl=95std**, **langlvl=90std**, or **langlvl=2003std** setting.

**Related information:** For more information about **NAMELIST** I/O, see *Namelist Formatting* in the *XL Fortran Language Reference*.

**nlwidth=record\_width**

By default, a **NAMELIST** write statement produces a single output record long enough to contain all of the written **NAMELIST** items. To restrict **NAMELIST** output records to a given width, use the **nlwidth** run-time option.

**Note:** The **RECL=** specifier for sequential files has largely made this option obsolete, because programs attempt to fit **NAMELIST** output within the specified record length. You can still use **nlwidth** in conjunction with **RECL=** as long as the **nlwidth** width does not exceed the stated record length for the file.

**random={generator1 | generator2}**

Specifies the generator to be used by `RANDOM_NUMBER` if `RANDOM_SEED` has not yet been called with the `GENERATOR` argument. The value `generator1` (the default) corresponds to `GENERATOR=1`, and `generator2` corresponds to `GENERATOR=2`. If you call `RANDOM_SEED` with the `GENERATOR` argument, it overrides the random option from that point onward in the program. Changing the random option by calling `SETRTEOPTS` after calling `RANDOM_SEED` with the `GENERATOR` option has no effect.

**scratch\_vars={yes | no}**

To give a specific name to a scratch file, set the `scratch_vars` run-time option to `yes`, and set the environment variable `XLFSRATCH_unit` to the name of the file you want to be associated with the specified unit number. See *Naming scratch files* in the *XL Fortran Optimization and Programming Guide* for examples.

**unit\_vars={yes | no}**

To give a specific name to an implicitly connected file or to a file opened with no `FILE=` specifier, you can set the run-time option `unit_vars=yes` and set one or more environment variables with names of the form `XLFUNIT_unit` to file names. See *Naming files that are connected with no explicit name* in the *XL Fortran Optimization and Programming Guide* for examples.

**uwidth={32 | 64}**

To specify the width of record length fields in unformatted sequential files, specify the value in bits. When the record length of an unformatted sequential file is greater than  $(2^{*31} - 1)$  bytes minus 8 bytes (for the record terminators surrounding the data), you need to set the run-time option `uwidth=64` to extend the record length fields to 64 bits. This allows the record length to be up to  $(2^{*63} - 1)$  minus 16 bytes (for the record terminators surrounding the data). The run-time option `uwidth` is only valid for 64-bit mode applications.

**xrf\_messages={yes | no}**

To prevent programs from displaying run-time messages for error conditions during I/O operations, `RANDOM_SEED` calls, and `ALLOCATE` or `DEALLOCATE` statements, set the `xrf_messages` run-time option to `no`. Otherwise, run-time messages for conversion errors and other problems are sent to the standard error stream.

The following examples set the `cnvrr` run-time option to `yes` and the `xrf_messages` option to `no`.

```
Basic format
XLFRTFOPTS=cnvrr=yes:xrf_messages=no
export XLFRTFOPTS

With imbedded blanks
XLFRTFOPTS="xrf_messages = NO : cnvrr = YES"
export XLFRTFOPTS
```

As a call to `SETRTEOPTS`, this example could be:

```
CALL setrteopts('xrf_messages=NO:cnvrr=yes')
! Name is in lowercase in case -U (mixed) option is used.
```

**Setting OMP and SMP run time options**

The `XLSMPOPTS` environment variable allows you to specify options that affect SMP execution. The OpenMP environment variables, `OMP_DYNAMIC`, `OMP_NESTED`, `OMP_NUM_THREADS`, and `OMP_SCHEDULE`, allow you to control the execution of parallel code. For details on using these, see `XLSMPOPTS` and OpenMP environment variables sections in the *XL Fortran Optimization and Programming Guide*.

## BLAS/ESSL environment variable

By default, the `libxlopt` library is linked with any application you compile with XL Fortran. However, if you are using a third-party Basic Linear Algebra Subprograms (BLAS) library or want to ship a binary that includes ESSL routines, you must specify these using the `XL_BLAS_LIB` environment variable. For example, if your own BLAS library is called `libblas`, set the environment variable as follows:

```
XL_BLAS_LIB="/usr/lib/libblas.a" export XL_BLAS_LIB
```

When the compiler generates calls to BLAS routines, the ones defined in the `libblas` library will be used at runtime instead of those defined in `libxlopt`.

## MALLOCALIGN=16

The `MALLOCALIGN=16` environment variable specifies that dynamic memory allocations return 16-byte aligned addresses. (See also the `-qipa=malloc16` option.)

## XL\_NOCLONEARCH

Use the `XL_NOCLONEARCH` to instruct the program to only execute the generic code, where generic code is the code that is not versioned for an architecture. The `XL_NOCLONEARCH` environment variable is not set by default; you can set it for debugging purposes in your application. (See also the `-qipa=clonearch` option.)

---

## Other environment variables that affect run-time behavior

The `LIBPATH` and `TMPDIR` environment variables have an effect at run time, as explained in “Correct settings for environment variables” on page 8. They are not XL Fortran run-time options and cannot be set in either `XLFRTEOPTS` or `XLSTMPOPTS`.

---

## XL Fortran run-time exceptions

The following operations cause run-time exceptions in the form of `SIGTRAP` signals, which typically result in a “Trace/BPT trap” message:

- Fixed-point division by zero.
- Character substring expression or array subscript out of bounds after you specified the `-C` option at compile time.
- Lengths of character pointer and target do not match after you specified the `-C` option at compile time.
- The flow of control in the program reaches a location for which a semantic error with severity of `S` was issued when the program was compiled.
- Floating-point exceptions occur after you specify the appropriate `-qflttrap` suboptions at compile time.
- Floating-point operations that generate NaN values and loads of the NaN values after you specify the `-qfloat=nanq` option at compile time.

If you install one of the predefined XL Fortran exception handlers before the exception occurs, a diagnostic message and a traceback showing the offset within each routine called that led to the exception are written to standard error after the exception occurs. The file buffers are also flushed before the program ends. If you compile the program with the `-g` option, the traceback shows source line numbers in addition to the address offsets.

You can use a symbolic debugger to determine the error. `dbx` provides a specific error message that describes the cause of the exception.

### Related information:

- “-C option” on page 89
- “-qflttrap option” on page 154
- “-qsigtrap option” on page 220

Also see the following topics in the *XL Fortran Optimization and Programming Guide*:

- *Detecting and trapping floating-point exceptions* for more details about these exceptions
- *Controlling the floating-point status and control register* for a list of exception handlers.

---

## Chapter 5. XL Fortran compiler-option reference

This section contains the following:

- Tables of compiler options. These tables are organized according to area of use and contain high-level information about the syntax and purpose of each option.
- Detailed information about each compiler option in “Detailed descriptions of the XL Fortran compiler options” on page 76.

---

### Summary of the XL Fortran compiler options

The following tables show the compiler options available in the XL Fortran compiler that you can enter in the configuration file, on the command line, or in the Fortran source code by using the **@PROCESS** compiler directive.

You can enter compiler options that start with **-q**, suboptions, and **@PROCESS** directives in either uppercase or lowercase. However, note that if you specify the **-qmixed** option, procedure names that you specify for the **-qextern** option are case-sensitive.

In general, this document uses the convention of lowercase for **-q** compiler options and suboptions and uppercase for **@PROCESS** directives. However, in the “Syntax” sections of this section and in the “Command-Line Option” column of the summary tables, we use uppercase letters in the names of **-q** options, suboptions, and **@PROCESS** directives to represent the minimum abbreviation for the keyword. For example, valid abbreviations for **-qOPTimize** are **-qopt**, **-qopti**, and so on.

Understanding the significance of the options you use and knowing the alternatives available can save you considerable time and effort in making your programs work correctly and efficiently.

## Options that control input to the compiler

The following options affect the compiler input at a high level. They determine which source files are processed and select case sensitivity, column sensitivity, and other global format issues.

**Related information:** See “XL Fortran input files” on page 23 and “Options that specify the locations of output files” on page 54.

Many of the options in “Options for compatibility” on page 64 change the permitted input format slightly.

Table 5. Options that control input to the compiler

| Command-Line Option  | @PROCESS Directive   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | See Page |
|----------------------|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -I <i>dir</i>        |                      | <p>Adds a directory to the search path for include files and <b>.mod</b> files. If XL Fortran calls <b>cpp</b>, this option adds a directory to the search path for <b>#include</b> files. Before checking the default directories for include and <b>.mod</b> files, the compiler checks each directory in the search path. For include files, this path is only used if the file name in an <b>INCLUDE</b> line is not provided with an absolute path. For <b>#include</b> files, refer to the <b>cpp</b> documentation for the details of the -I option.</p> <p><b>Default:</b> The following directories are searched, in the following order:</p> <ol style="list-style-type: none"> <li>1. The current directory</li> <li>2. The directory where the source file is</li> <li>3. <b>/usr/include</b>.</li> </ol> | 95       |
| -qci= <i>numbers</i> | CI( <i>numbers</i> ) | <p>Activates the specified <b>INCLUDE</b> lines.</p> <p><b>Default:</b> No default value.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | 128      |
| -qcr<br>-qnocr       |                      | <p>Allows you to control how the compiler interprets the CR (carriage return) character. This allows you to compile code written using a Mac OS or DOS/Windows editor.</p> <p><b>Default:</b> -qcr</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 130      |

Table 5. Options that control input to the compiler (continued)

| Command-Line Option                         | @PROCESS Directive                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | See Page |
|---------------------------------------------|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qdirective<br>[= <i>directive_list</i> ]   | DIRECTIVE<br>[[ <i>directive_list</i> ]]   | Specifies sequences of characters, known as trigger constants, that identify comment lines as compiler comment directives.<br><b>Default:</b> Comment lines beginning with <b>IBM*</b> are considered directives. If you specify <b>-qsmp=omp</b> , only <b>\$OMP</b> is considered to be a directive trigger. All other directive triggers are turned off unless you explicitly turn them back on. If you specify <b>-qsmp=noomp</b> (noomp is the default for -qsmp), <b>IBMP</b> , <b>\$OMP</b> and <b>SMP\$</b> are considered directive triggers, along with any other directive triggers that are turned on (such as <b>IBM*</b> and <b>IBMT</b> ). If you have also specified <b>-qthreaded</b> , comment lines beginning with <b>IBMT</b> are also considered directives. | 135      |
| -qnodirective<br>[= <i>directive_list</i> ] | NODIRECTIVE<br>[[ <i>directive_list</i> ]] |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |          |
| -qfixed<br>[= <i>right_margin</i> ]         | FIXED<br>[[ <i>right_margin</i> ]]         | Indicates that the input source program is in fixed source form and optionally specifies the maximum line length.<br><b>Default:</b> <b>-qfree=f90</b> for the <b>f90</b> , <b>xl f90</b> , <b>xl f90_r</b> , <b>xl f90_r7</b> , <b>f95</b> , <b>xl f95</b> , <b>xl f95_r</b> , and <b>xl f95_r7</b> commands and <b>-qfixed=72</b> for the <b>xl f</b> , <b>xl f_r</b> , <b>xl f_r7</b> , and <b>f77/fort77</b> commands.                                                                                                                                                                                                                                                                                                                                                        | 150      |
| -qfree[={f90 ibm}]<br>-k                    | FREE[({F90   IBM})]                        | Indicates that the source code is in free form. The <b>ibm</b> and <b>f90</b> suboptions specify compatibility with the free source form defined for VS FORTRAN and Fortran 90/Fortran 95, respectively. <b>-k</b> and <b>-qfree</b> are short forms for <b>-qfree=f90</b> .<br><b>Default:</b> <b>-qfree=f90</b> for the <b>f90</b> , <b>f95</b> , <b>xl f90</b> , <b>xl f90_r</b> , <b>xl f90_r7</b> , <b>xl f95</b> , <b>xl f95_r</b> , and <b>xl f95_r7</b> commands and <b>-qfixed=72</b> for the <b>xl f</b> , <b>xl f_r</b> , <b>xl f_r7</b> , and <b>f77/fort77</b> commands.                                                                                                                                                                                             | 156      |
| -qmbsc<br>-qnombsc                          | MBCS<br>NOMBCS                             | Indicates to the compiler whether character literal constants, Hollerith constants, H edit descriptors, and character string edit descriptors can contain Multibyte Character Set (MBCS) or Unicode characters.<br><b>Default:</b> <b>-qnombsc</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 189      |
| -U<br>-qmixed<br>-qnomixed                  | MIXED<br>NOMIXED                           | Makes the compiler sensitive to the case of letters in names.<br><b>Default:</b> <b>-qnomixed</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 262      |

Table 5. Options that control input to the compiler (continued)

| Command-Line Option   | @PROCESS Directive | Description                                           | See Page |
|-----------------------|--------------------|-------------------------------------------------------|----------|
| -qsuffix={suboptions} |                    | Specifies the source-file suffix on the command line. | 233      |

## Options that specify the locations of output files

The following options specify names or directories where the compiler stores output files.

In the table, an \* indicates that the option is processed by the **ld** command, rather than by the XL Fortran compiler; you can find more information about these options in the AIX information for the **ld** command.

**Related information:** See “XL Fortran Output files” on page 25 and “Options that control input to the compiler” on page 52.

Table 6. Options that specify the locations of output files

| Command-Line Option        | @PROCESS Directive | Description                                                                                                                                                     | See Page |
|----------------------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -d                         |                    | Leaves preprocessed source files produced by <b>cpp</b> , instead of deleting them.<br><b>Default:</b> Temporary files produced by <b>cpp</b> are deleted.      | 92       |
| -o <i>name</i> *           |                    | Specifies a name for the output object, executable, or assembler source file.<br><b>Default:</b> -o a.out                                                       | 102      |
| -qmoddir= <i>directory</i> |                    | Specifies the location for any module ( <b>.mod</b> ) files that the compiler writes.<br><b>Default:</b> <b>.mod</b> files are placed in the current directory. | 191      |

## Options for performance optimization

The following options can help you to speed up the execution of your XL Fortran programs or to find areas of poor performance that can then be tuned. The most important such option is **-O**. In general, the other performance-related options work much better in combination with **-O**; some have no effect at all without **-O**.

**Related information:** See *Optimizing XL compiler programs* in the *XL Fortran Optimization and Programming Guide*.

Some of the options in “Options for floating-point processing” on page 71 can also improve performance, but you must use them with care to avoid error conditions and incorrect results.

Table 7. Options for performance optimization

| Command-Line Option                                               | @PROCESS Directive                     | Description                                                                                                                           | See Page |
|-------------------------------------------------------------------|----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|----------|
| -O[ <i>level</i> ]<br>-qoptimize[= <i>level</i> ]<br>-qnooptimize | OPTimize[ <i>level</i> ]<br>NOOPTimize | Specifies whether code is optimized during compilation and, if so, at which level (0, 2, 3, 4, or 5).<br><b>Default:</b> -qnooptimize | 100      |

Table 7. Options for performance optimization (continued)

| Command-Line Option                                                          | @PROCESS Directive                                                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | See Page |
|------------------------------------------------------------------------------|------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -p<br>-pg                                                                    |                                                                              | Sets up the object file for profiling.<br><b>Default:</b> No profiling.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 103      |
| -Q<br>-Q!<br>-Q+ <i>names</i><br>-Q- <i>names</i>                            |                                                                              | Specifies whether procedures are inlined and/or particular procedures that should or should not be inlined. <i>names</i> is a list of procedure names separated by colons.<br><b>Default:</b> No inlining.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 104      |
| -qalias=<br>{[no]aryovrlp  <br>[no]intptr  <br>[no]pteovrlp  <br>[no]std...} | ALIAS(<br>{[NO]ARYOVRLP<br>  [NO]INTPTR<br>  [NO]PTEOVRLP<br>  [NO]STD)... ) | Indicates whether a program contains certain categories of aliasing. The compiler limits the scope of some optimizations when there is a possibility that different names are aliases for the same storage locations.<br><b>Default:</b> -<br><b>qalias=aryovrlp:nointptr:pteovrlp:std</b> for the <b>xl</b> f90, <b>xl</b> f90_r7, <b>xl</b> f95, <b>xl</b> f95_r, <b>xl</b> f95_r7, <b>f</b> 90, and <b>f</b> 95 commands; <b>-qalias=aryovrlp:intptr:pteovrlp:std</b> for the <b>xl</b> f, <b>xl</b> f_r, <b>xl</b> _r7, <b>f</b> 77, and <b>fort</b> 77, commands.                                                                                                                                                                                                                                                                     | 109      |
| -qalign={[no]4k  <br>struct {=subopt}  <br>bindc {=subopt}}                  | ALIGN( {[NO]4K  <br>STRUCT({subopt})  <br>BINDC({subopt}) )                  | Specifies the alignment of data objects in storage, which avoids performance problems with misaligned data. The <b>[no]4k</b> , <b>bindc</b> , and <b>struct</b> options can be specified and are not mutually exclusive. The <b>[no]4k</b> option is useful primarily in combination with logical volume I/O and disk striping.<br><b>Default:</b> -<br><b>qalign=no4k:struct=natural:bindc=power</b>                                                                                                                                                                                                                                                                                                                                                                                                                                     | 112      |
| -qarch=<br><i>architecture</i>                                               |                                                                              | Controls which instructions the compiler can generate. Changing the default can improve performance but might produce code that can only be run on specific machines. The choices are <b>auto</b> , <b>com</b> , <b>pwr</b> , <b>pwr2</b> (or <b>pwr</b> x), <b>pwr2s</b> , <b>p2sc</b> , <b>601</b> , <b>603</b> , <b>604</b> , <b>ppc</b> , <b>ppcgr</b> , <b>ppc64</b> , <b>ppc64gr</b> , <b>ppc64grsq</b> , <b>rs64a</b> , <b>rs64b</b> , <b>rs64c</b> , <b>pwr3</b> , <b>pwr4</b> , <b>pwr5</b> , <b>pwr5x</b> , <b>ppc64v</b> , and <b>ppc970</b> .<br><b>Default:</b> <b>-qarch=com</b> if you specify <b>-q32</b> , which uses only instructions that are common to all platforms. If you specify <b>-q64</b> , the default is <b>ppc64</b> , which allows you to run the executable file on any 64-bit PowerPC hardware platform. | 114      |
| -qassert={deps  <br>nodeps  <br>itercnt= <i>n</i> }                          | ASSERT(DEPS  <br>NODEPS  <br>ITERCNT( <i>N</i> ))                            | Provides information about the characteristics of the files that can help to fine-tune optimizations.<br><b>Default:</b> <b>-qassert=deps:itercnt=1024</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 119      |

Table 7. Options for performance optimization (continued)

| Command-Line Option                                                                                                                                                              | @PROCESS Directive                 | Description                                                                                                                                                                                                                                                                                                                                                                                                              | See Page |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qcache={<br>auto  <br>assoc= <i>number</i>  <br>cost= <i>cycles</i>  <br>level= <i>level</i>  <br>line= <i>bytes</i>  <br>size= <i>Kbytes</i>  <br>type={C c D d <br>I i}[:...] |                                    | Specifies the cache configuration for a specific execution machine. The compiler uses this information to tune program performance, especially for loop operations that can be structured (or <i>blocked</i> ) to process only the amount of data that can fit into the data cache.<br><b>Default:</b> The compiler uses typical values based on the <b>-qtune</b> setting, the <b>-qarch</b> setting, or both settings. | 124      |
| -qcompact<br>-qnocompact                                                                                                                                                         | COMPACT<br>NOCOMPACT               | Reduces optimizations that increase code size.<br><b>Default:</b> <b>-qnocompact</b>                                                                                                                                                                                                                                                                                                                                     | 129      |
| -qdirectstorage<br>-qnodirectstorage                                                                                                                                             |                                    | Informs the compiler that a given compilation unit may reference write-through-enabled or cache-inhibited storage. Use this option with discretion. It is intended for programmers who know how the memory and cache blocks work, and how to tune their applications for optimal performance.<br><b>Default:</b> <b>-qnodirectstorage</b>                                                                                | 137      |
| -qenablevmx<br>-qnoenablevmx                                                                                                                                                     |                                    | Enables the generation of VMX (Vector Multimedia eXtension) instructions.<br><b>Default:</b> <b>-qnoenablevmx</b>                                                                                                                                                                                                                                                                                                        | 141      |
| -qessl                                                                                                                                                                           |                                    | Allows the use of Engineering and Scientific Subroutine Library (ESSL) routines in place of Fortran 90 Intrinsic Procedures. Use the ESSL Serial Library when linking with <b>-lessl</b> . Use the ESSL SMP Library when linking with <b>-lesslmp</b> .<br><b>Default:</b> <b>-qnoessl</b>                                                                                                                               | 144      |
| -qfdpr<br>-qnofdpr                                                                                                                                                               |                                    | Provides object files with information needed for the AIX fdpr (Feedback Directed Program Restructuring) performance-tuning utility to optimize the resulting executable file.<br><b>Default:</b> <b>-qnofdpr</b>                                                                                                                                                                                                        | 149      |
| -qhot[= <i>suboptions</i> ]<br>-qnohot                                                                                                                                           | HOT[= <i>suboptions</i> ]<br>NOHOT | Instructs the compiler to perform high-order loop analysis and transformations during optimization.<br><b>Default:</b> <b>-qnohot</b>                                                                                                                                                                                                                                                                                    | 159      |
| -qipa[= <i>suboptions</i> ]<br>  -qnoipa                                                                                                                                         |                                    | Enhances <b>-O</b> optimization by doing detailed analysis across procedures (interprocedural analysis or IPA).<br><b>Default:</b> <b>-O</b> analyzes each subprogram separately, ruling out certain optimizations that apply across subprogram boundaries. Note that specifying <b>-O5</b> is equivalent to specifying <b>-O4</b> and <b>-qipa=level=2</b> .                                                            | 169      |

Table 7. Options for performance optimization (continued)

| Command-Line Option                                                   | @PROCESS Directive   | Description                                                                                                                                                                                                                                                                                                                    | See Page |
|-----------------------------------------------------------------------|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qkeepparm<br>-qnokeepparm                                            |                      | Ensures that incoming procedure parameters are stored on the stack even when optimizing.<br><b>Default: -qnokeepparm</b>                                                                                                                                                                                                       | 176      |
| -qlargepage<br>-qnolargepage                                          |                      | Indicates to the compiler that a program, designed to execute in a large page memory environment, can take advantage of large 16 MB pages provided on POWER4 and higher based systems.<br><b>Default: -qnolargepage</b>                                                                                                        | 179      |
| -qmaxmem=<br>Kbytes                                                   | MAXMEM<br>(Kbytes)   | Limits the amount of memory that the compiler allocates while performing specific, memory-intensive optimizations to the specified number of kilobytes. A value of -1 allows optimization to take as much memory as it needs without checking for limits.<br><b>Default: -qmaxmem=8192; At -O3, -O4, and -O5, -qmaxmem=-1.</b> | 187      |
| -qpdf{1 2}                                                            |                      | Tunes optimizations through <i>profile-directed feedback</i> (PDF), where results from sample program execution are used to improve optimization near conditional branches and in frequently executed code sections.<br><b>Default: Optimizations use fixed assumptions about branch frequency and other statistics.</b>       | 198      |
| -qprefetch  <br>-qnoprefetch                                          |                      | Indicates whether or not the prefetch instructions should be inserted automatically by the compiler.<br><b>Default: -qprefetch</b>                                                                                                                                                                                             | 208      |
| -qshowpdf<br>-qnoshowpdf                                              |                      | Adds additional call and block count profiling information to an executable. This option is used together with the <b>-qpdf1</b> option.<br><b>Default: -qnoshowpdf</b>                                                                                                                                                        | 219      |
| -qsmallstack[=<br>dynlenonheap  <br>nodynlenonheap]<br>-qnosmallstack |                      | Specifies that the compiler will minimize stack usage where possible.<br><b>Default: -qnosmallstack</b>                                                                                                                                                                                                                        | 221      |
| -qsmp[= <i>suboptions</i> ]<br>-qnosmp                                |                      | When used with <b>xlf_r</b> , <b>xlf_r7</b> , <b>xlf90_r</b> , <b>xlf90_r7</b> , <b>xlf95_r</b> , or <b>xlf95_r7</b> , controls automatic parallelization of loops, user-directed parallelization of loops and other items, and the choice of chunking algorithm.<br><b>Default: -qnosmp</b>                                   | 222      |
| -NSbytes<br>-qSPILLsize=<br>bytes                                     | SPILLsize<br>(bytes) | Specifies the size of the register spill space.<br><b>Default: -NS512</b>                                                                                                                                                                                                                                                      | 99       |

Table 7. Options for performance optimization (continued)

| Command-Line Option                                    | @PROCESS Directive                                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | See Page |
|--------------------------------------------------------|------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qstacktemp={<br>0   -1  <br><i>positive_integer</i> } | STACKTEMP={<br>0   -1  <br><i>positive_integer</i> } | Determines where to allocate applicable XL Fortran compiler temporaries at run time.<br><b>Default:</b> 0 The compiler allocates applicable temporaries on the heap or the stack at its discretion.                                                                                                                                                                                                                                                                               | 229      |
| -qstrict<br>-qnostrict                                 | STRICT<br>NOSTRICT                                   | Ensures that optimizations done by the -O3, -O4, and -O5 options do not alter the semantics of a program.<br><b>Default:</b> With -O3 and higher levels of optimization in effect, code may be rearranged so that results or exceptions are different from those in unoptimized programs. For -O2, the default is -qstrict. This option is ignored for -qnoopt.                                                                                                                   | 230      |
| -qstrict_induction<br>-qnostrict_induction             |                                                      | Prevents the compiler from performing induction (loop counter) variable optimizations. These optimizations may be <i>unsafe</i> (may alter the semantics of your program) when there are integer overflow operations involving the induction variables.<br><b>Default:</b> -qnostrict_induction                                                                                                                                                                                   | 232      |
| -qthreaded                                             |                                                      | Specifies that the compiler should generate thread-safe code. This is turned on by default for the xlf_r, xlf_r7, xlf90_r, xlf90_r7, xlf95_r, and xlf95_r7 commands.                                                                                                                                                                                                                                                                                                              | 239      |
| -qtune=<br><i>implementation</i>                       |                                                      | Tunes instruction selection, scheduling, and other implementation-dependent performance enhancements for a specific implementation of a hardware architecture. The following settings are valid: auto, pwr, pwr2 (or pwrx), pwr2s, p2sc, 601, 603, 604, rs64a, rs64b, rs64c, pwr3, pwr4, pwr5, or ppc970.<br><b>Default:</b> -qtune=pwr2, if you specify -q32 and enable the -qarch=com option. If you specify -q64 and enable the -qarch=ppc option, the default is -qtune=pwr3. | 240      |
| -qunroll[=auto  <br>yes]<br>-qnounroll                 |                                                      | Specifies whether the compiler is allowed to automatically unroll DO loops.<br><b>Default:</b> -qunroll=auto                                                                                                                                                                                                                                                                                                                                                                      | 244      |
| -qunwind<br>-qnounwind                                 | UNWIND<br>NOUNWIND                                   | Specifies that the compiler will preserve the default behavior for saves and restores to volatile registers during a procedure call.<br><b>Default:</b> -qunwind                                                                                                                                                                                                                                                                                                                  | 245      |
| -qvecnvolf<br>-qnovecnvol                              |                                                      | Specifies whether to use volatile or non-volatile vector registers.<br><b>Default:</b> -qnovecnvol                                                                                                                                                                                                                                                                                                                                                                                | 246      |

Table 7. Options for performance optimization (continued)

| Command-Line Option        | @PROCESS Directive     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | See Page |
|----------------------------|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qzerosize<br>-qnozerosize | ZEROSIZE<br>NOZEROSIZE | Improves performance of FORTRAN 77 and some Fortran 90 and Fortran 95 programs by preventing checking for zero-sized character strings and arrays.<br><b>Default:</b> <b>-qzerosize</b> for the <b>xlF90</b> , <b>xlF90_r</b> , <b>xlF90_r7</b> , <b>xlF95</b> , <b>xlF95_r</b> , <b>xlF95_r7</b> , <b>f90</b> , and <b>f95</b> commands and <b>-qnozerosize</b> for the <b>xlF</b> , <b>xlF_r</b> , <b>xlF_r7</b> , <b>f77</b> , and <b>fort77</b> commands (meaning these commands cannot be used for programs that contain zero-sized objects). | 259      |

## Options for error checking and debugging

The following options help you avoid, detect, and correct problems in your XL Fortran programs and can save you having to refer as frequently to Chapter 7, “Problem determination and debugging,” on page 275.

In particular, **-qlanglvl** helps detect portability problems early in the compilation process by warning of potential violations of the Fortran standards. These can be due to extensions in the program or due to compiler options that allow such extensions.

Other options, such as **-C** and **-qfltrap**, detect and/or prevent run-time errors in calculations, which could otherwise produce incorrect output.

Because these options require additional checking at compile time and some of them introduce run-time error checking that slows execution, you may need to experiment to find the right balance between extra checking and slower compilation and execution performance.

Using these options can help to minimize the amount of problem determination and debugging you have to do. Other options you may find useful while debugging include:

- “-# option” on page 77, “-v option” on page 264, and “-V option” on page 265
- “-qalias option” on page 109
- “-qci option” on page 128
- “-qobject option” on page 195
- “-qreport option” on page 213
- “-qsource option” on page 227

Table 8. Options for debugging and error checking

| Command-Line Option        | @PROCESS Directive | Description                                                                                                                                                                                                                                                            | See Page |
|----------------------------|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -C<br>-qcheck<br>-qnocheck | CHECK<br>NOCHECK   | Checks each reference to an array element, array section, or character substring for correctness. Out-of-bounds references are reported as severe errors if found at compile time and generate <b>SIGTRAP</b> signals at run time.<br><b>Default:</b> <b>-qnocheck</b> | 89       |

Table 8. Options for debugging and error checking (continued)

| Command-Line Option                                | @PROCESS Directive                             | Description                                                                                                                                                                                                                                                                                                                                                                  | See Page |
|----------------------------------------------------|------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -D<br>-qdlines<br>-qnodlines                       | DLINES<br>NODLINES                             | Specifies whether fixed source form lines with a D in column 1 are compiled or treated as comments.<br><b>Default: -qnodlines</b>                                                                                                                                                                                                                                            | 91       |
| -g<br>-qdbg<br>-qnodbg                             | DBG<br>NODBG                                   | Generates debug information for use by a symbolic debugger.<br><b>Default: -qnodbg</b>                                                                                                                                                                                                                                                                                       | 94       |
| -qdpcl<br>-qnodpcl                                 | DPCL<br>NODPCL                                 | Generates symbols that tools based on the Dynamic Probe Class Library (DPCL) can use to see the structure of an executable file.<br><b>Default: -qnodpcl</b>                                                                                                                                                                                                                 | 140      |
| -qextchk<br>-qnoextchk                             | EXTCHK<br>NOEXTCHK                             | Sets up type-checking information for common blocks, procedure definitions, procedure references, and module data. Later, the linker can detect mismatches across compilation units by using this information.<br><b>Default: -qnoextchk</b>                                                                                                                                 | 145      |
| -qflttrap<br>[= <i>suboptions</i> ]<br>-qnoflttrap | FLTTRAP<br>[= <i>suboptions</i> ]<br>NOFLTTRAP | Determines what types of floating-point exception conditions to detect at run time. The program receives a SIGTRAP signal when the corresponding exception occurs.<br><b>Default: -qnoflttrap</b>                                                                                                                                                                            | 154      |
| -qfullpath<br>-qnofullpath                         |                                                | Records the full, or absolute, path names of source and include files in object files compiled with debugging information (-g option).<br><b>Default:</b> The relative path names of source files are recorded in the object files.                                                                                                                                          | 157      |
| -qhalt= <i>sev</i>                                 | HALT( <i>sev</i> )                             | Stops before producing any object, executable, or assembler source files if the maximum severity of compile-time messages equals or exceeds the specified severity. <i>severity</i> is one of i, l, w, e, s, u, or q, meaning informational, language, warning, error, severe error, unrecoverable error, or a severity indicating “don’t stop”.<br><b>Default: -qhalt=S</b> | 158      |

Table 8. Options for debugging and error checking (continued)

| Command-Line Option                                                                                                     | @PROCESS Directive                                                                                                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | See Page |
|-------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qinitauto[= <i>hex_value</i> ]<br>-qnoinitauto                                                                         |                                                                                                                        | Initializes each byte or word (4 bytes) of storage for automatic variables to a specific value, depending on the length of the <i>hex_value</i> . This helps you to locate variables that are referenced before being defined. For example, by using both the <b>-qinitauto</b> option to initialize <b>REAL</b> variables with a NaNs value and the <b>-qfltrap</b> option, it is possible to identify references to uninitialized <b>REAL</b> variables at run time.<br><b>Default:</b> <b>-qnoinitauto</b> . If you specify <b>-qinitauto</b> without a <i>hex_value</i> , the compiler initializes the value of each byte of automatic storage to zero. | 164      |
| -qlanglvl={<br>77std<br>  90std<br>  90pure<br>  90ext<br>  95std<br>  95pure<br>  2003std<br>  2003pure<br>  extended} | LANGLVL({<br>77STD<br>  90STD<br>  90PURE<br>  90EXT<br>  95STD<br>  95PURE<br>  2003STD<br>  2003PURE<br>  EXTENDED}) | Determines which language standard (or superset, or subset of a standard) to consult for nonconformance. It identifies nonconforming source code and also options that allow such nonconformances.<br><b>Default:</b> <b>-qlanglvl=extended</b>                                                                                                                                                                                                                                                                                                                                                                                                             | 177      |
| -qlinedebug<br>-qnoinedebug                                                                                             | LINEDEBUG<br>NOLINEDEBUG                                                                                               | Generates line number and source file name information for the debugger.<br><b>Default:</b> <b>-qnoinedebug</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 183      |
| -qOBJECT<br>-qNOOBJECT                                                                                                  | OBJECT<br>NOOBJECT                                                                                                     | Specifies whether to produce an object file or to stop immediately after checking the syntax of the source files.<br><b>Default:</b> <b>-qobject</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 195      |
| -qsaa<br>-qnosaa                                                                                                        | SAA<br>NOSAA                                                                                                           | Checks for conformance to the SAA FORTRAN language definition. It identifies nonconforming source code and also options that allow such nonconformances.<br><b>Default:</b> <b>-qnosaa</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 215      |
| -qsaveopt<br>-qnosaveopt                                                                                                |                                                                                                                        | Saves the command-line options used for compiling a source file in the corresponding object file.<br><b>Default:</b> <b>-qnosaveopt</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 217      |
| -qsigtrap[=<br><i>trap_handler</i> ]                                                                                    |                                                                                                                        | Installs <code>xl__trce</code> or a predefined or user-written trap handler in a main program.<br><b>Default:</b> No trap handler installed; program core dumps when a <b>trap</b> instruction is executed.                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 220      |

Table 8. Options for debugging and error checking (continued)

| Command-Line Option                   | @PROCESS Directive | Description                                                                                                                                                                                                                                                                                                                            | See Page |
|---------------------------------------|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qtbtable={none<br>  small<br>  full} |                    | Limits the amount of debugging traceback information in object files, to reduce the size of the program.<br><b>Default:</b> Full traceback information in the object file when compiling non-optimized (without <b>-O</b> ) or for debugging (with <b>-g</b> ). Otherwise, a small amount of traceback information in the object file. | 238      |
| -qwarn64<br>-qnowarn64                |                    | Detects the truncation of an 8-byte integer pointer to 4 bytes. Identifies, through informational messages, statements that might cause problems during 32-bit to 64-bit migration.<br><b>Default:</b> <b>-qnowarn64</b>                                                                                                               | 248      |
| -qxflag=dvz                           |                    | Specifying <b>-qxflag=dvz</b> causes the compiler to generate code to detect floating-point divide-by-zero operations.<br><b>Default:</b> No code is generated to detect floating-point divide-by-zero operations.                                                                                                                     | 249      |
| -qxlines<br>-qnoxlines                | XLINES<br>NOXLINES | Specifies whether fixed source form lines with a X in column 1 are treated as source code and compiled, or treated instead as comments.<br><b>Default:</b> <b>-qnoxlines</b>                                                                                                                                                           | 256      |

## Options that control listings and messages

The following options determine whether the compiler produces a listing (.lst file), what kinds of information go into the listing, and what the compiler does about any error conditions it detects.

Some of the options in “Options for error checking and debugging” on page 59 can also produce compiler messages.

Table 9. Options that control listings and messages

| Command-Line Option       | @PROCESS Directive   | Description                                                                                                                                                        | See Page |
|---------------------------|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -#                        |                      | Generates information on the progress of the compilation without actually running the individual components.<br><b>Default:</b> No progress messages are produced. | 77       |
| -qattr[=full]<br>-qnoattr | ATTR[FULL]<br>NOATTR | Specifies whether to produce the attribute component of the attribute and cross-reference section of the listing.<br><b>Default:</b> <b>-qnoattr</b>               | 120      |

Table 9. Options that control listings and messages (continued)

| Command-Line Option                                                                     | @PROCESS Directive                                                | Description                                                                                                                                                                                                                                                                                                                         | See Page |
|-----------------------------------------------------------------------------------------|-------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qflag=<br><i>listing_severity</i> :<br><i>terminal_severity</i><br>-w                  | FLAG<br>( <i>listing_severity</i> ,<br><i>terminal_severity</i> ) | Limits the diagnostic messages to those of a specified level or higher. Only messages with severity <i>listing_severity</i> or higher are written to the listing file. Only messages with severity <i>terminal_severity</i> or higher are written to the terminal. -w is a short form for -qflag=e:e.<br><b>Default: -qflag=i:i</b> | 151      |
| -qlist[={no}offset]<br>-qnolist                                                         | LIST[([NO]OFFSET)]<br>NOLIST                                      | Specifies whether to produce the object section of the listing.<br><b>Default: -qnolist</b>                                                                                                                                                                                                                                         | 184      |
| -qlistopt<br>-qnolistopt                                                                | LISTOPT<br>NOLISTOPT                                              | Determines whether to show the setting of every compiler option in the listing file or only selected options. These selected options include those specified on the command line or directives plus some that are always put in the listing.<br><b>Default: -qnolistopt</b>                                                         | 185      |
| -qnoprint                                                                               |                                                                   | Prevents the listing file from being created, regardless of the settings of other listing options.<br><b>Default:</b> A listing is produced if you specify any of -qattr, -qlist, -qlistopt, -qphsinfo, -qreport, -qsource, or -qxref.                                                                                              | 193      |
| -qphsinfo<br>-qnophsinfo                                                                | PHSINFO<br>NOPHSINFO                                              | Determines whether timing information is displayed on the terminal for each compiler phase.<br><b>Default: -qnophsinfo</b>                                                                                                                                                                                                          | 202      |
| -qreport[={smp<br>  hotlist}...]<br>-qnoreport                                          | REPORT<br>[([SMPLIST  <br>HOTLIST}...)]<br>NOREPORT               | Determines whether to produce transformation reports showing how the program is parallelized and how loops are optimized.<br><b>Default: -qnoreport</b>                                                                                                                                                                             | 213      |
| -qsource<br>-qnosource                                                                  | SOURCE<br>NOSOURCE                                                | Determines whether to produce the source section of the listing.<br><b>Default: -qnosource</b>                                                                                                                                                                                                                                      | 227      |
| -qsuppress<br>[= <i>nnnn-mmm</i> [: <i>nnnn-mmm</i> ...]  <br>cmpmsg]  <br>-qnosuppress |                                                                   | Specifies which messages to suppress from the output stream.                                                                                                                                                                                                                                                                        | 234      |
| -qversion<br>-qnoverion                                                                 |                                                                   | Displays the version and release of the compiler being invoked.<br><b>Default: -qnoverion</b>                                                                                                                                                                                                                                       | 247      |
| -qxref<br>-qnoxref<br>-qxref=full                                                       | XREF<br>NOXREF<br>XREF(FULL)                                      | Determines whether to produce the cross-reference component of the attribute and cross-reference section of the listing.<br><b>Default: -qnoxref</b>                                                                                                                                                                                | 258      |

Table 9. Options that control listings and messages (continued)

| Command-Line Option | @PROCESS Directive | Description                                                                                                                                                                                                                                                 | See Page |
|---------------------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -S                  |                    | Produces one or more <code>.s</code> files showing equivalent assembler source for each Fortran source file.<br><b>Default:</b> No equivalent assembler source is produced.                                                                                 | 260      |
| -v                  |                    | Traces the progress of the compilation by displaying the name and parameters of each compiler component that is executed by the invocation command.<br><b>Default:</b> No progress messages are produced.                                                   | 264      |
| -V                  |                    | Traces the progress of the compilation by displaying the name and parameters of each compiler component that is executed by the invocation command. These are displayed in a shell-executable format.<br><b>Default:</b> No progress messages are produced. | 265      |

## Options for compatibility

The following options help you maintain compatibility between your XL Fortran source code on past, current, and future hardware platforms or help you port programs to XL Fortran with a minimum of changes.

**Related information:** *Porting programs to XL Fortran* in the *XL Fortran Optimization and Programming Guide* discusses this subject in more detail. *Duplicating the floating-point results of other systems* in the *XL Fortran Optimization and Programming Guide* explains how to use some of the options in “Options for floating-point processing” on page 71 to achieve floating-point results compatible with other systems.

The `-qfree=ibm` form of the “-qfree option” on page 156 also provides compatibility with VS FORTRAN free source form.

Table 10. Options for compatibility

| Command-Line Option                                                       | @PROCESS Directive                           | Description                                                                                                                                                                                                                                                                                                                                                                | See Page |
|---------------------------------------------------------------------------|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qautodbl= <i>setting</i>                                                 | AUTODBL( <i>setting</i> )                    | Provides an automatic means of converting single-precision floating-point calculations to double-precision and of converting double-precision calculations to extended-precision. Use one of the following settings: none, dbl, dbl4, dbl8, dblpad, dblpad4, or dblpad8.<br><b>Default:</b> -qautodbl=none                                                                 | 121      |
| -qcclines<br>-qnocclines                                                  | CCLINES<br>NOCCLINES                         | Determines whether the compiler recognizes conditional compilation lines.<br><b>Default:</b> -qcclines if you have specified the -qsmp=omp option; otherwise, -qnocclines.                                                                                                                                                                                                 | 126      |
| -qctyp <sub>lss</sub><br>[=( <i>[no]</i> arg)]<br>-qnoctyp <sub>lss</sub> | CTYPLSS<br>[[ <i>[NO]</i> ARG)]<br>NOCTYPLSS | Specifies whether character constant expressions are allowed wherever typeless constants may be used. This language extension might be needed when you are porting programs from other platforms. Suboption <i>arg</i> specifies that Hollerith constants used as actual arguments will be treated as integer actual arguments.<br><b>Default:</b> -qnoctyp <sub>lss</sub> | 131      |
| -qddim<br>-qnoddim                                                        | DDIM<br>NODDIM                               | Specifies that the bounds of pointee arrays are re-evaluated each time the arrays are referenced and removes some restrictions on the bounds expressions for pointee arrays.<br><b>Default:</b> -qnoddim                                                                                                                                                                   | 134      |

Table 10. Options for compatibility (continued)

| Command-Line Option         | @PROCESS Directive     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | See Page |
|-----------------------------|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qdpc<br>-qdpc=e<br>-qnodpc | DPC<br>DPC(E)<br>NODPC | Increases the precision of real constants, for maximum accuracy when assigning real constants to <b>DOUBLE PRECISION</b> variables. This language extension might be needed when you are porting programs from other platforms.<br><b>Default: -qnodpc</b>                                                                                                                                                                                                                                                                                                    | 139      |
| -qescape<br>-qnoescape      | ESCAPE<br>NOESCAPE     | Specifies how the backslash is treated in character strings, Hollerith constants, H edit descriptors, and character string edit descriptors. It can be treated as an escape character or as a backslash character. This language extension might be needed when you are porting programs from other platforms.<br><b>Default: -qescape</b>                                                                                                                                                                                                                    | 143      |
| -qextern= <i>names</i>      |                        | Allows user-written procedures to be called instead of XL Fortran intrinsics. <i>names</i> is a list of procedure names separated by colons. The procedure names are treated as if they appear in an <b>EXTERNAL</b> statement in each compilation unit being compiled. If any of your procedure names conflict with XL Fortran intrinsic procedures, use this option to call the procedures in the source code instead of the intrinsic ones.<br><b>Default:</b> Names of intrinsic procedures override user-written procedure names when they are the same. | 146      |

Table 10. Options for compatibility (continued)

| Command-Line Option                              | @PROCESS Directive                            | Description                                                                                                                                                                                                                                                                                                   | See Page |
|--------------------------------------------------|-----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qextname[= <i>name:name...</i> ]<br>-qnoextname | EXTNAME[( <i>name:name...</i> )]<br>NOEXTNAME | Adds an underscore to the names of all global entities, which helps in porting programs from systems where this is a convention for mixed-language programs.<br><b>Default:</b> -qnoextname                                                                                                                   | 147      |
| -qinit=f90ptr                                    | INIT(f90ptr)                                  | Makes the initial association status of pointers disassociated.<br><b>Default:</b> The default association status of pointers is undefined.                                                                                                                                                                   | 163      |
| -qintlog<br>-qnointlog                           | INTLOG<br>NOINTLOG                            | Specifies that you can mix integer and logical values in expressions and statements.<br><b>Default:</b> -qnointlog                                                                                                                                                                                            | 166      |
| -qintsize= <i>bytes</i>                          | INTSIZE( <i>bytes</i> )                       | Sets the size of default <b>INTEGER</b> and <b>LOGICAL</b> values.<br><b>Default:</b> -qintsize=4                                                                                                                                                                                                             | 167      |
| -qlog4<br>-qnolog4                               | LOG4<br>NOLOG4                                | Specifies whether the result of a logical operation with logical operands is a <b>LOGICAL(4)</b> or is a <b>LOGICAL</b> with the maximum length of the operands.<br><b>Default:</b> -qnolog4                                                                                                                  | 186      |
| -qmodule=mangle81                                |                                               | Specifies that the compiler should use the XL Fortran Version 8.1 naming convention for non-intrinsic module files.<br><b>Default:</b> The compiler uses the current naming convention for non-intrinsic module names. This convention is not compatible with that used by previous versions of the compiler. | 192      |

Table 10. Options for compatibility (continued)

| Command-Line Option                           | @PROCESS Directive                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | See Page |
|-----------------------------------------------|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qnullterm<br>-qnonnullterm                   | NULLTERM<br>NONULLTERM                       | Appends a null character to each character constant expression that is passed as a dummy argument, to make it more convenient to pass strings to C functions.<br><b>Default: -qnonnullterm</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 194      |
| -1<br>-qonetrip<br>-qnoonetrip                | ONETRIP<br>NOONETRIP                         | Executes each <b>DO</b> loop in the compiled program at least once if its <b>DO</b> statement is executed, even if the iteration count is 0.<br><b>Default: -qnoonetrip</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 78       |
| -qpport<br>[=suboptions]<br>-qnoport          | PORT<br>[=suboptions]<br>NOPORT              | Increases flexibility when porting programs to XL Fortran by providing a number of options to accommodate other Fortran language extensions.<br><b>Default: -qnoport</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 205      |
| -qposition=<br>{appendold  <br>appendunknown} | POSITION(<br>{APPENDOLD  <br>APPENDUNKNOWN}) | Positions the file pointer at the end of the file when data is written after an <b>OPEN</b> statement with no <b>POSITION=</b> specifier and the corresponding <b>STATUS=</b> value ( <b>OLD</b> or <b>UNKNOWN</b> ) is specified.<br><b>Default:</b> Depends on the I/O specifiers in the <b>OPEN</b> statement and on the compiler invocation command:<br><b>-qposition=appendold</b> for the <b>xl</b> , <b>xl_r</b> , <b>xl_r7</b> , and <b>f77/fort77</b> commands and the defined Fortran 90 and Fortran 95 behaviors for the <b>xl90</b> , <b>xl90_r</b> , <b>xl90_r7</b> , <b>xl95</b> , <b>xl95_r</b> , <b>xl95_r7</b> , <b>f90</b> and <b>f95</b> commands. | 207      |

Table 10. Options for compatibility (continued)

| Command-Line Option                      | @PROCESS Directive                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                             | See Page |
|------------------------------------------|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qqcount<br>-qnoqcount                   | QCOUNT<br>NOQCOUNT                  | Accepts the <b>Q</b> character-count edit descriptor ( <b>Q</b> ) as well as the extended-precision <b>Q</b> edit descriptor ( <b>Qw.d</b> ). With -qnoqcount, all <b>Q</b> edit descriptors are interpreted as the extended-precision <b>Q</b> edit descriptor.<br><b>Default: -qnoqcount</b>                                                                                                                                          | 209      |
| -qrealize= <i>bytes</i>                  | REALSIZE( <i>bytes</i> )            | Sets the default size of <b>REAL</b> , <b>DOUBLE PRECISION</b> , <b>COMPLEX</b> , and <b>DOUBLE COMPLEX</b> values.<br><b>Default: -qrealize=4</b>                                                                                                                                                                                                                                                                                      | 210      |
| -qsave[={all   defaultinit}]<br>-qnosave | SAVE{(ALL   DEFAULTINIT)}<br>NOSAVE | Specifies the default storage class for local variables. <b>-qsave</b> , <b>-qsave=all</b> , or <b>-qsave=defaultinit</b> sets the default storage class to <b>STATIC</b> , while <b>-qnosave</b> sets it to <b>AUTOMATIC</b> .<br><b>Default: -qnosave</b><br><br><b>-qsave</b> is turned on by default for <b>xlf</b> , <b>xlf_r</b> , <b>xlf_r7</b> , <b>f77</b> , or <b>fort77</b> to duplicate the behavior of FORTRAN77 commands. | 216      |
| -qslk=[centi   micro ]                   |                                     | Specifies that when returning a value using the <b>SYSTEM_CLOCK</b> intrinsic procedure, the compiler will use centisecond resolution. You can specify a microsecond resolution by using <b>-qslk=micro</b> .<br><br><b>Default: -qslk=centi</b>                                                                                                                                                                                        | 218      |
| -qswapomp<br>-qnoswapomp                 | SWAPOMP<br>NOSWAPOMP                | Specifies that the compiler should recognize and substitute OpenMP routines in XL Fortran programs.<br><b>Default: -qswapomp</b>                                                                                                                                                                                                                                                                                                        | 236      |

Table 10. Options for compatibility (continued)

| Command-Line Option        | @PROCESS Directive | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | See Page |
|----------------------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -u<br>-qundef<br>-qnoundef | UNDEF<br>NOUNDEF   | Specifies whether implicit typing of variable names is permitted. -u and -qundef have the same effect as the <b>IMPLICIT NONE</b> statement that appears in each scope that allows implicit statements.<br><b>Default: -qnoundef</b>                                                                                                                                                                                                                                                                      | 263      |
| -qxflag=oldtab             | XFLAG(OLDTAB)      | Interprets a tab in columns 1 to 5 as a single character (for fixed source form programs), for compatibility with XL Fortran Version 1.<br><b>Default:</b> Tab is interpreted as one or more characters.                                                                                                                                                                                                                                                                                                  | 250      |
| -qxlf77=settings           | XLF77(settings)    | Provides backward compatibility with XL Fortran Versions 1 and 2 aspects of language semantics and I/O data format that have changed. Most of these changes are required by the Fortran 90 standard.<br><b>Default:</b> Default suboptions are blankpad, nogedit77, nointarg, nointxor, leadzero, nooldboz, nopersistent, and nosofteof for the xlf90, xlf90_r, xlf90_r7, xlf95, xlf95_r, xlf95_r7, f90, and f95 commands and are the exact opposite for the xlf, xlf_r, xlf_r7, and f77/fort77 commands. | 252      |

Table 10. Options for compatibility (continued)

| Command-Line Option                               | @PROCESS Directive                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                       | See Page |
|---------------------------------------------------|--------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qxlf90=<br>{[no]signedzero  <br>[no]autodealloc} | XLF90(<br>{[no]signedzero  <br>[no]autodealloc}) | Determines whether the compiler provides the Fortran 90 or the Fortran 95 level of support for certain aspects of the language.<br><b>Default:</b> The default suboptions are <b>signedzero</b> and <b>autodealloc</b> for the <b>xlF95</b> , <b>xlF95_r</b> , <b>xlF95_r7</b> , and <b>f95</b> invocation commands. For all other invocation commands, the default suboptions are <b>nosignedzero</b> and <b>noautodealloc</b> . | 254      |

## Options for floating-point processing

To take maximum advantage of the system floating-point performance and precision, you may need to specify details of how the compiler and XLF-compiled programs perform floating-point calculations.

**Related information:** See “-qflttrap option” on page 154 and *Duplicating the floating-point results of other systems* in the *XL Fortran Optimization and Programming Guide*.

Table 11. Options for floating-point processing

| Command-Line Option                                          | @PROCESS Directive                          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                      | See Page |
|--------------------------------------------------------------|---------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qfloat=options                                              | FLOAT(options)                              | Determines how the compiler generates or optimizes code to handle particular types of floating-point calculations.<br><b>Default:</b> Default suboptions are <b>nofltint</b> , <b>fold</b> , <b>nohsflt</b> , <b>nohssngl</b> , <b>maf</b> , <b>nonans</b> , <b>norndsngl</b> , <b>norm</b> , <b>norsqrt</b> , and <b>nostrictnmaf</b> ; some of these settings are different with <b>-O3</b> optimization turned on or with <b>-qarch=ppc</b> . | 152      |
| -qieee={ Near<br>  Minus<br>  Plus<br>  Zero}<br>-y{n m p z} | IEEE({Near<br>  Minus<br>  Plus<br>  Zero}) | Specifies the rounding mode for the compiler to use when evaluating constant floating-point expressions at compile time.<br><b>Default:</b> <b>-qieee=near</b>                                                                                                                                                                                                                                                                                   | 162      |
| -qstrictieemod<br>-qnostrictieemod                           | STRICTIEEE-<br>MOD<br>NOSTRICTIEEE-<br>MOD  | Specifies whether the compiler will adhere to the Fortran 2003 IEEE arithmetic rules for the <b>ieee_arithmetic</b> and <b>ieee_exceptions</b> intrinsic modules.<br><b>Default:</b> <b>-qstrictieemod</b>                                                                                                                                                                                                                                       | 231      |

## Options that control linking

The following options control the way the **ld** command processes object files during compilation. Some of these options are passed on to **ld** and are not processed by the compiler at all.

You can actually include **ld** options on the compiler command line, because the compiler passes unrecognized options on to the linker.

In the table, an \* indicates that the option is processed by the **ld** command, rather than the XL Fortran compiler; you can find more information about these options in the AIX information for the **ld** command.

**Related information:** The “-qextchk option” on page 145 enables some extra consistency checking during linking.

Other linker options you might find helpful are the following:

- **-brename** (to change individual symbol names to avoid unresolved references)
- **-bmap** (to produce a map file showing information such as sizes of common blocks)

Table 12. Options that control linking

| Command-Line Option | @PROCESS Directive | Description                                                                                                                                                                                                                                                                                     | See Page |
|---------------------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -b64*               |                    | Instructs <b>ld</b> to bind 64-bit objects in 64-bit mode.                                                                                                                                                                                                                                      | 80       |
| -bdynamic*          |                    | These options are toggles used to control the processing of <b>-l</b> options and the way that shared objects are processed.                                                                                                                                                                    | 81       |
| -bshared*           |                    |                                                                                                                                                                                                                                                                                                 |          |
| -bstatic*           |                    |                                                                                                                                                                                                                                                                                                 |          |
| -bhalt:error_level* |                    | Specifies the maximum error level allowed before linker command processing halts.<br><b>Default:</b> <b>-bhalt:4</b> , as specified in the configuration file.                                                                                                                                  | 83       |
| -bloadmap:name*     |                    | Requests that a log of linker actions and messages be saved in file <i>name</i> .<br><b>Default:</b> No log is kept.                                                                                                                                                                            | 84       |
| -bmaxdata:bytes*    |                    | Specifies the maximum amount of space to reserve for the program data segment and stack segment for programs where the size of these regions is a constraint.<br><b>Default:</b> Combined stack and data space is slightly less than 256 MB, or lower, depending on the limits for the user ID. | 85       |
| -bmaxstack:bytes*   |                    |                                                                                                                                                                                                                                                                                                 |          |
| -brtl*              |                    | Determines which algorithm is used to find libraries (specified with the <b>-l</b> option).                                                                                                                                                                                                     | 86       |
| -bnortl*            |                    |                                                                                                                                                                                                                                                                                                 |          |
| -c                  |                    | Produces an object file instead of an executable file.<br><b>Default:</b> Compile and link-edit, producing an executable file.                                                                                                                                                                  | 90       |
| -Ldir*              |                    | Looks in the specified directory for libraries specified by the <b>-l</b> option.<br><b>Default:</b> <b>/usr/lib</b>                                                                                                                                                                            | 97       |
| -lkey*              |                    | Searches the specified library file, where <i>key</i> selects the file <b>libkey.a</b> .<br><b>Default:</b> Libraries listed in <b>xlf.cfg</b> .                                                                                                                                                | 98       |

Table 12. Options that control linking (continued)

| Command-Line Option    | @PROCESS Directive | Description                                                                                                      | See Page |
|------------------------|--------------------|------------------------------------------------------------------------------------------------------------------|----------|
| -qp[ic]=large   small] |                    | Generates Position Independent Code (PIC) that can be used in shared libraries.<br><b>Default:</b> -qp[ic]=small | 204      |

## Options that Control Other Compiler Operations

These options can help to do the following:

- Control internal size limits for the compiler
- Determine names and options for commands that are executed during compilation
- Determine the bit mode and instruction set for the target architecture

Table 13. Options that control the compiler internal operation

| Command-Line Option                                 | @PROCESS Directive | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | See Page |
|-----------------------------------------------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -Bprefix                                            |                    | Determines a substitute path name for executable files used during compilation, such as the compiler or linker. It can be used in combination with the -t option, which determines which of these components are affected by -B.<br><b>Default:</b> Paths for these components are defined in the configuration file, the \$PATH environment variable, or both.                                                                                                                         | 79       |
| -Fconfig_file<br>-Fconfig_file: stanza<br>-F:stanza |                    | Specifies an alternative configuration file, the stanza to use within the configuration file, or both.<br><b>Default:</b> The configuration file is /etc/xlf.cfg, and the stanza depends on the name of the command that executes the compiler.                                                                                                                                                                                                                                         | 93       |
| -q32                                                |                    | Sets the bit mode and instruction set for a 32-bit target architecture.                                                                                                                                                                                                                                                                                                                                                                                                                 | 105      |
| -q64                                                |                    | Sets the bit mode and instruction set for a 64-bit target architecture.                                                                                                                                                                                                                                                                                                                                                                                                                 | 106      |
| -tcomponents                                        |                    | Applies the prefix specified by the -B option to the designated components. <i>components</i> can be one or more of p, F, c, d, I, a, h, b, z, or l, with no separators, corresponding to an optimizing preprocessor, the C preprocessor, the compiler, the -S disassembler, the interprocedural analysis (IPA) tool, the assembler, the loop optimizer, the code generator, the binder, and the linker, respectively.<br><b>Default:</b> -B prefix, if any, applies to all components. | 261      |

Table 13. Options that control the compiler internal operation (continued)

| Command-Line Option | @PROCESS Directive | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | See Page |
|---------------------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -Wcomponent,options |                    | <p>Passes the listed options to a component that is executed during compilation. <i>component</i> is p, F, c, d, I, a, z, or l, corresponding to an optimizing preprocessor, the C preprocessor, the compiler, the -S disassembler, the interprocedural analysis (IPA) tool, the assembler, the binder, and the linker, respectively.</p> <p><b>Default:</b> The options passed to these programs are as follows:</p> <ul style="list-style-type: none"> <li>• Those listed in the configuration file</li> <li>• Any unrecognized options on the command line (passed to the linker)</li> </ul> | 266      |

## Options that are obsolete or not recommended

The following options are obsolete for either or both of the following reasons:

- It has been replaced by an alternative that is considered to be better. Usually this happens when a limited or special-purpose option is replaced by one with a more general purpose and additional features.
- We expect that few or no customers use the feature and that it can be removed from the product in the future with minimal impact to current users.

### Notes:

1. If you do use any of these options in existing makefiles or compilation scripts, you should migrate to the new alternatives as soon as you can to avoid any potential problems in the future.
2. The **append** suboption of **-qposition** has been replaced by **appendunknown**.

Table 14. Options that are obsolete or not recommended

| Command-Line Option  | @PROCESS Directive  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | See Page |
|----------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -P{v k}[!]           |                     | <p>Obsolete. It is still accepted, but it has no effect. Invokes the selected optimizing preprocessor. Adding ! prevents the compilation step from following preprocessing.</p> <p><b>Note:</b> The preprocessors are available as separate vendor-logo products.</p> <p><b>Default:</b> No preprocessing.</p>                                                                                                                                                              |          |
| -qcharlen=length     | CHARLEN<br>(length) | <p>Obsolete. It is still accepted, but it has no effect. The maximum length for character constants and subobjects of constants is 32 767 bytes (32 KB). The maximum length for character variables is 268 435 456 bytes (256 MB) in 32-bit mode. The maximum length for character variables is 2**40 bytes in 64-bit mode. These limits are always in effect and are intended to be high enough to avoid portability problems with programs that contain long strings.</p> |          |
| -qhsflt<br>-qnohsflt | HSFLT<br>NOHSFLT    | <p>Obsolete. Replaced by the <b>hsflt</b> and <b>nohsflt</b> suboptions of the “-qfloat option” on page 152.</p>                                                                                                                                                                                                                                                                                                                                                            |          |
| -qhssngl<br>-qnssngl | HSSNGL<br>NOHSSNGL  | <p>Obsolete. Replaced by the <b>hssngl</b> and <b>nohssngl</b> suboptions of the “-qfloat option” on page 152.</p>                                                                                                                                                                                                                                                                                                                                                          |          |

Table 14. Options that are obsolete or not recommended (continued)

| Command-Line Option  | @PROCESS Directive | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | See Page |
|----------------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| -qlm<br>-qnolm       |                    | Obsolete. The license management control system (LM) is no longer supported in this product version.                                                                                                                                                                                                                                                                                                                                                                                             |          |
| -qrecur<br>-qnorecur | RECUR<br>NORECUR   | <p>Not recommended. Specifies whether external subprograms may be called recursively.</p> <p>For new programs, use the <b>RECURSIVE</b> keyword, which provides a standard-conforming way of using recursive procedures. If you specify the -qrecur option, the compiler must assume that any procedure could be recursive. Code generation for recursive procedures may be less efficient. Using the <b>RECURSIVE</b> keyword allows you to specify exactly which procedures are recursive.</p> | 212      |

---

## Detailed descriptions of the XL Fortran compiler options

The following alphabetical list of options provides all the information you should need to use each option effectively.

How to read the syntax information:

- Syntax is shown first in command-line form, and then in **@PROCESS** form if applicable.
- Defaults for each option are underlined and in boldface type.
- 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 (...).

## **-# option**

### **Syntax**

`-#`

Generates information on the progress of the compilation without actually running the individual components.

### **Rules**

At the points where the compiler executes commands to perform different compilation steps, this option displays a simulation of the system calls it would do and the system argument lists it would pass, but it does not actually perform these actions.

Examining the output of this option can help you quickly and safely determine the following information for a particular compilation:

- What files are involved
- What options are in effect for each step

It avoids the overhead of compiling the source code and avoids overwriting any existing files, such as `.lst` files. (If you are familiar with the `make` command, it is similar to `make -n`.)

Note that if you specify this option with `-qipa`, the compiler does not display linker information subsequent to the IPA link step. This is because the compiler does not actually call IPA.

### **Related information**

The “-v option” on page 264 and “-V option” on page 265 produce the same output but also performs the compilation.

## **-1 option**

### **Syntax**

-1  
ONETRIP | NOONETRIP

Executes each **DO** loop in the compiled program at least once if its **DO** statement is executed, even if the iteration count is 0. This option provides compatibility with FORTRAN 66. The default is to follow the behavior of later Fortran standards, where **DO** loops are not performed if the iteration count is 0.

### **Restrictions**

It has no effect on **FORALL** statements, **FORALL** constructs, or array constructor implied-**DO** loops.

### **Related information**

-qonetrip is the long form of -1.

## **-B option**

### **Syntax**

*-Bprefix*

Determines a substitute path name for executable files used during compilation, such as the compiler or linker. It can be used in combination with the **-t** option, which determines which of these components are affected by **-B**.

### **Arguments**

*prefix* is the name of a directory where the alternative executable files reside. It must end in a / (slash).

### **Rules**

To form the complete path name for each component, the driver program adds *prefix* to the standard program names. You can restrict the components that are affected by this option by also including one or more **-t***mnemonic* options.

You can also specify default path names for these commands in the configuration file.

This option allows you to keep multiple levels of some or all of the XL Fortran components or to try out an upgraded component before installing it permanently. When keeping multiple levels of XL Fortran available, you might want to put the appropriate **-B** and **-t** options into a configuration-file stanza and to use the **-F** option to select the stanza to use.

### **Examples**

In this example, an earlier level of the XL Fortran components is installed in the directory **/usr/lpp/xlf/bin**. To test the upgraded product before making it available to everyone, the system administrator restores the latest install image under the directory **/home/jim** and then tries it out with commands similar to:

```
xlf95 -tchbI -B/home/jim/usr/lpp/xlf/bin/ test_suite.f
```

Once the upgrade meets the acceptance criteria, the system administrator installs it over the old level in **/usr/lpp/xlf/bin**.

### **Related information**

See “**-t** option” on page 261, “**-F** option” on page 93, “Customizing the configuration file” on page 11, and “Running two levels of XL Fortran” on page 18.

## **-b64 option**

### **Syntax**

`-b64`

The AIX operating system provides 64-bit shared object files in both **libc.a** and **libm.a**. In 64-bit mode, you can use the **-b64** linker option to instruct **ld** to bind with 64-bit objects.

### **Related information**

For more information on the 64-bit environment, see Chapter 6, “Using XL Fortran in a 64-Bit Environment,” on page 271. For more information on **-b64**, see *AIX General Programming Concepts*.

## -bdynamic, -bshared, and -bstatic Options

### Syntax

**-bdynamic** | **-bshared** | **-bstatic**

These options are toggles that are used to control the processing of **-l** options and the way that shared objects are processed.

The options **-bdynamic** and **-bshared** are synonymous.

When **-bstatic** is in effect, shared objects are statically linked into the output file. When **-bdynamic** is in effect, shared objects are linked dynamically.

When **-brtl** is used in conjunction with either **-bdynamic** or **-bshared**, the search for libraries specified with the **-l** option is satisfied by the suffix *.so* or *.a*. For each directory searched, a file with the suffix *.so* is looked for. If it is not found, a file with the suffix *.a* is looked for. If neither file is found, the search continues with the next directory.

### Rules

These options are passed directly to the **ld** command and are not processed by XL Fortran at all.

These options are position-significant. They affect all files that are specified after the option on the command-line.

Table 15 summarizes how these options interact with **-brtl** and **-bnortl** to affect the file suffix that is being searched.

Table 15. Interaction of new linker options

|                  |                             | Position-significant                          |                 |
|------------------|-----------------------------|-----------------------------------------------|-----------------|
|                  |                             | <b>-bdynamic</b><br><b>-bshared (default)</b> | <b>-bstatic</b> |
| Global Influence | <b>-brtl</b>                | <i>.so</i><br><i>.a</i>                       | <i>.a</i>       |
|                  | <b>-bnortl</b><br>(default) | <i>.a</i>                                     | <i>.a</i>       |

## Examples

```
xlf95 f.f -brtl -bshared -lmylib
```

In this case, the linker searches for the library **libmylib.so** first and then the library **libmylib.a** in each directory in the search path consecutively until either is encountered.

```
xlf95_r f.f -bdynamic -llib1 -bstatic -llib2 -brtl
```

In this case, to satisfy the first library specification, the linker searches for the library **liblib1.so** first and then the library **liblib1.a** in each directory (as described in the previous example). However, at the same time the linker only searches for **liblib2.a** in those same directories.

## Related information

For more information on these options, see *AIX General Programming Concepts*. See also “-brtl option” on page 86.

## **-bhalt option**

### **Syntax**

`-bhalt:error_level`

Specifies the maximum error level that is allowed before the linker (**ld**) command halts. The default value is 4, as specified in the configuration file. If any linker command has an error return value greater than the value that is specified by the *error\_level* variable, linking stops.

### **Rules**

This option is passed directly to the **ld** command and is not processed by XL Fortran at all.

## **-bloadmap option**

### **Syntax**

`-bloadmap:name`

Requests that a log of linker actions and messages be saved in file *name*. You can use the log to help diagnose linking problems. For example, the log contains information about type mismatches that the **-qextchk** option detected.

### **Rules**

This option is passed directly to the **ld** command and is not processed by XL Fortran at all.

## -bmaxdata, -bmaxstack Options

### Syntax

```
-bmaxdata:bytes
-bmaxstack:bytes
```

Specifies the maximum amount of space to reserve for the program data segment and stack segment for programs where the size of these regions is a constraint.

### Background information

The data segment holds, among other things, heap storage that is used by the program.

If your program allocates large arrays, statically or dynamically, specify **-bmaxdata** when linking the program. The resulting executable program uses the large data model and can have a data region larger than a single segment, up to a maximum of 2 GB. Refer to the **ld** documentation in the *AIX Commands Reference* for allowable values. Note that since the compiler might create temporary arrays during compilation, it may be useful to define a value for the **-bmaxdata** compiler option in anticipation of this.

If the program has large amounts of automatic data or otherwise exceeds the soft limit on stack size for a program, specify **-bmaxstack** when you link the program. Use this option to define the soft limit up to 256 MB for 32-bit mode or up to the limit imposed by system resources for 64-bit mode. However, each main program or subprogram is limited to 256 MB per instance.

### Arguments

You can specify the size as a decimal, octal (which is prefixed by **0**), or hexadecimal value (which is prefixed by **0x**).

### Rules

These options are passed directly to the **ld** command and are not processed by XL Fortran at all.

### Examples

```
xlf95 -O3 -qhot -bmaxdata:0x20000000 huge_data_model.f
xlf95 -O3 -qhot -bmaxstack:2000000 lots_of_automatic_data.f
```

### Related information

For a discussion of the issues involved in creating large AIX programs, see “Large Program Support Overview” in *AIX General Programming Concepts*.

## **-brtl option**

### **Syntax**

`-brtl` | `-bnortl`

Determines which algorithm will be used to find libraries that are specified with the `-l` option.

### **Background information**

If `-brtl` is specified, run-time linking is enabled.

When used in conjunction with either `-bdynamic` or `-bshared`, the search for libraries that you specified with the `-l` option is satisfied by the suffix `.so` or `.a`. For each directory searched, a file with the suffix `.so` is looked for. If it is not found, a file with the suffix `.a` is looked for. If neither file is found, the search continues with the next directory. Table 15 on page 81 gives a graphical representation of how these options combine to affect the file suffix being searched for.

### **Rules**

These options are passed directly to the `ld` command and are not processed by XL Fortran at all. Only the last specified of these options will be used. These options have a global effect; regardless of where they appear on the command line, they affect the entire command.

### **Examples**

```
xlf95 -brtl f.f -lmylib
xlf95_r -bnortl f.f -bdynamic -llib1 -bstatic -llib2
```

Note that if you add `-brtl` to the end of the last example, it will override the earlier occurrence of `-bnortl`.

### **Related information**

For more information on these options, see *AIX General Programming Concepts*. See also “`-bdynamic`, `-bshared`, and `-bstatic` Options” on page 81.

## **-bshared option**

### **Related information**

See “-bdynamic, -bshared, and -bstatic Options” on page 81.

## **-bstatic option**

### **Related information**

See “-bdynamic, -bshared, and -bstatic Options” on page 81.

## -C option

### Syntax

-C  
CHECK | NOCHECK

Checks each reference to an array element, array section, or character substring for correctness.

### Rules

At compile time, if the compiler can determine that a reference goes out of bounds, the severity of the error reported is increased to **S** (severe) when this option is specified.

At run time, if a reference goes out of bounds, the program generates a **SIGTRAP** signal. By default, this signal ends the program and produces a core dump. This is expected behavior and does not indicate there is a defect in the compiler product.

Because the run-time checking can slow execution, you should decide which is the more important factor for each program: the performance impact or the possibility of incorrect results if an error goes undetected. You might decide to use this option only while testing and debugging a program (if performance is more important) or also for compiling the production version (if safety is more important).

### Related information

The **-C** option prevents some of the optimizations that the “-qhot option” on page 159 performs. You may want to remove the **-C** option after debugging of your code is complete and to add the **-qhot** option to achieve a more thorough optimization.

The valid bounds for character substring expressions differ depending on the setting of the **-qzerosize** option. See “-qzerosize option” on page 259.

“-qsigtrap option” on page 220 and *Installing an exception handler* in the *XL Fortran Optimization and Programming Guide* describe how to detect and recover from **SIGTRAP** signals without ending the program.

**-qcheck** is the long form of **-C**.

## **-c option**

### **Syntax**

`-c`

Prevents the completed object file from being sent to the **ld** command for link-editing. With this option, the output is a **.o** file for each source file.

Using the **-o** option in combination with **-c** selects a different name for the **.o** file. In this case, you can only compile one source file at a time.

### **Related information**

See “**-o option**” on page 102.

## **-D option**

### **Syntax**

`-D`  
`DLINES` | `NODLINES`

Specifies whether the compiler compiles fixed source form lines with a **D** in column 1 or treats them as comments.

If you specify `-D`, the fixed source form lines that have a **D** in column 1 are compiled. The default action is to treat these lines as comment lines. They are typically used for sections of debugging code that need to be turned on and off.

Note that in order to pass C-style `-D` macro definitions to the C preprocessor, for example, compiling a file that ends with `.F`, use the `-W` option. For example:

```
-WF, -DDEFINE_THIS
```

### **Related information**

`-qdlines` is the long form of `-D`.

## **-d option**

### **Syntax**

-d

Causes preprocessed source files that are produced by **cpp** to be kept rather than to be deleted.

### **Rules**

The files that this option produces have names of the form *Ffilename.f*, derived from the names of the original source files.

### **Related information**

See “Passing Fortran files through the C preprocessor” on page 29.

## **-F option**

### **Syntax**

```
-F{config_file | config_file:stanza | :stanza}
```

Specifies an alternative configuration file, which stanza to use within the configuration file, or both.

The configuration file specifies different kinds of defaults, such as options for particular compilation steps and the locations of various files that the compiler requires. A default configuration file (*/etc/xlf.cfg*) is supplied at installation time. The default stanza depends on the name of the command used to invoke the compiler (*xlf90*, *xlf90\_r*, *xlf90\_r7*, *xlf95*, *xlf95\_r*, *xlf95\_r7*, *xlf*, *xlf\_r*, *xlf\_r7*, *f77*, or *fort77*).

A simple way to customize the way the compiler works, as an alternative to writing complicated compilation scripts, is to add new stanzas to */etc/xlf.cfg*, giving each stanza a different name and a different set of default compiler options. You may find the single, centralized file easier to maintain than many scattered compilation scripts and makefiles.

By running the compiler with an appropriate **-F** option, you can select the set of options that you want. You might have one set for full optimization, another set for full error checking, and so on.

### **Restrictions**

Because the default configuration file is replaced each time a new compiler release is installed, make sure to save a copy of any new stanzas or compiler options that you add.

### **Examples**

```
Use stanza debug in default xlf.cfg.
xlf95 -F:debug t.f
```

```
Use stanza xlf95 in /home/fred/xlf.cfg.
xlf95 -F/home/fred/xlf.cfg t.f
```

```
Use stanza myxlf in /home/fred/xlf.cfg.
xlf95 -F/home/fred/xlf.cfg:myxlf t.f
```

### **Related information**

“Customizing the configuration file” on page 11 explains the contents of a configuration file and tells how to select different stanzas in the file without using the **-F** option.

## **-g option**

### **Syntax**

-g  
DBG | **NODBG**

Generates debug information for use by a symbolic debugger.

**-qdbg** is the long form of **-g**.

**-g** implies the **-Q!** option.

### **Related information**

- “-qlinedebug option” on page 183
- “Debugging a Fortran 90 or Fortran 95 program” on page 282
- “A Sample dbx session for an XL Fortran program” on page 282
- “Symbolic debugger support” on page 5

## -I option

### Syntax

*-I**dir*

Adds a directory to the search path for include files and **.mod** files. If XL Fortran calls **cpp**, this option adds a directory to the search path for **#include** files. Before checking the default directories for include and **.mod** files, the compiler checks each directory in the search path. For include files, this path is only used if the file name in an **INCLUDE** line is not provided with an absolute path. For **#include** files, refer to the **cpp** documentation for the details of the **-I** option.

### Arguments

*dir* must be a valid path name (for example, **/home/dir**, **/tmp**, or **./subdir**).

### Rules

The compiler appends a **/** to the *dir* and then concatenates that with the file name before making a search. If you specify more than one **-I** option on the command line, files are searched in the order of the *dir* names as they appear on the command line.

The following directories are searched, in this order, after any paths that are specified by **-I** options:

1. The current directory (from which the compiler is executed)
2. The directory where the source file is (if different from 1)
3. **/usr/include**.

### Related information

The “-qmoddir option” on page 191 puts **.mod** files in a specific directory when you compile a file that contains modules.

## **-k option**

### **Syntax**

-k  
FREE(F90)

Specifies that the program is in free source form.

### **Applicable product levels**

The meaning of this option has changed from XL Fortran Version 2. To get the old behavior of **-k**, use the option **-qfree=ibm** instead.

### **Related information**

See “-qfree option” on page 156 and *Free Source Form* in the *XL Fortran Language Reference*.

This option is the short form of **-qfree=f90**.

## **-L option**

### **Syntax**

*-Ldir*

Looks in the specified directory for libraries that are specified by the **-l** option. If you use libraries other than the default ones in **/usr/lib**, you can specify one or more **-L** options that point to the locations of the other libraries. You can also set the **LIBPATH** environment variable, which lets you specify a search path for libraries at run time.

### **Rules**

This option is passed directly to the **ld** command and is not processed by XL Fortran at all.

### **Related information**

See “Options that control linking” on page 72 and “Linking XL Fortran programs” on page 31.

## **-l option**

### **Syntax**

`-lkey`

Searches the specified library file, where *key* selects the library **libkey.a**.

### **Rules**

This option is passed directly to the **ld** command and is not processed by XL Fortran at all.

### **Related information**

See “Options that control linking” on page 72 and “Linking XL Fortran programs” on page 31.

## **-NS option**

### **Syntax**

*-NSbytes*  
*SPILLSIZE(bytes)*

Specifies the size (in bytes) of the register spill space; the internal program storage areas used by the optimizer for register spills to storage.

### **Rules**

It defines the number of bytes of stack space to reserve in each subprogram, in case there are too many variables to hold in registers and the program needs temporary storage for register contents.

### **Defaults**

By default, each subprogram stack has 512 bytes of spill space reserved.

If you need this option, a compile-time message informs you of the fact.

### **Related information**

*-qspillsize* is the long form of *-NS*.

## -O option

### Syntax

`-O[level]`  
`OPTimize[(level)] | NOOPTimize`

Specifies whether to optimize code during compilation and, if so, at which level:

### Arguments

#### not specified

Almost all optimizations are disabled. (This is equivalent to specifying **-O0** or **-qnoopt**.)

- O** For each release of XL Fortran, **-O** enables the level of optimization that represents the best tradeoff between compilation speed and run-time performance. If you need a specific level of optimization, specify the appropriate numeric value. Currently, **-O** is equivalent to **-O2**.
- O0** Almost all optimizations are disabled. This option is equivalent to **-qnoopt**.
- O1** Reserved for future use. This form does not currently do any optimization and is ignored. In past releases, it was interpreted as a combination of the **-O** and **-1** options, which may have had unintended results.
- O2** Performs a set of optimizations that are intended to offer improved performance without an unreasonable increase in time or storage that is required for compilation.
- O3** Performs additional optimizations that are memory intensive, compile-time intensive, and may change the semantics of the program slightly, unless **-qstrict** is specified. We recommend these optimizations when the desire for run-time speed improvements outweighs the concern for limiting compile-time resources.

This level of optimization also affects the setting of the **-qfloat** option, turning on the **fltint** and **rsqrt** suboptions by default, and sets **-qmaxmem=-1**.

Specifying **-O3** implies **-qhot=level=0** unless you explicitly specify **-qhot** or **-qhot=level=1**.

- O4** Aggressively optimizes the source program, trading off additional compile time for potential improvements in the generated code. You can specify the option at compile time or at link time. If you specify it at link time, it will have no effect unless you also specify it at compile time for at least the file that contains the main program.

**-O4** implies the following other options:

- **-qhot**
- **-qipa**
- **-O3** (and all the options and settings that it implies)
- **-qarch=auto**
- **-qtune=auto**
- **-qcache=auto**

Note that the **auto** setting of **-qarch**, **-qtune**, and **-qcache** implies that the execution environment will be the same as the compilation environment.

This option follows the "last option wins" conflict resolution rule, so any of the options that are modified by **-O4** can be subsequently changed. For example, specifying **-O4 -qarch=com** allows aggressive intraprocedural optimization while maintaining code portability.

- O5** Provides all of the functionality of the **-O4** option, but also provides the functionality of the **-qipa=level=2** option.

**Note:** Combining **-O2** and higher optimizations with **-qsmp=omp** invokes additional optimization algorithms, including interprocedural analysis (IPA). IPA optimizations provide opportunities for the compiler to generate additional **fmadd** instructions.

To obtain the same floating-point accuracy for optimized and non-optimized applications, you must specify the **-qfloat=nomaf** compiler option. In cases where differences in floating-point accuracy still occur after specifying **-qfloat=nomaf**, the **-qstrict** compiler option allows you to exert greater control over changes that optimization can cause in floating-point semantics.

## Restrictions

Generally, use the same optimization level for both the compile and link steps. This is important when using either the **-O4** or **-O5** optimization level to get the best run-time performance. For the **-O5** level, all loop transformations (as specified via the **-qhot** option) are done at the link step.

Increasing the level of optimization may or may not result in additional performance improvements, depending on whether the additional analysis detects any further optimization opportunities.

An optimization level of **-O3** or higher can change the behavior of the program and potentially cause exceptions that would not otherwise occur. Use of the **-qstrict** option can eliminate potential changes and exceptions.

If the **-O** option is used in an **@PROCESS** statement, only an optimization level of 0, 2, or 3 is allowed.

Compilations with optimization may require more time and machine resources than other compilations.

The more the compiler optimizes a program, the more difficult it is to debug the program with a symbolic debugger.

## Related information

"-qessl Option" on page 144 allows the use of ESSL routines.

"-qstrict option" on page 230 shows how to turn off the effects of **-O3** that might change the semantics of a program.

"-qipa option" on page 169, "-qhot option" on page 159, and "-qpdf option" on page 198 turn on additional optimizations that may improve performance for some programs.

*Optimizing XL compiler programs* in the *XL Fortran Optimization and Programming Guide* discusses technical details of the optimization techniques the compiler uses and some strategies you can use to get maximum performance from your code.

**-qOPTimize** is the long form of **-O**.

## **-o option**

### **Syntax**

*-o name*

Specifies a name for the output object, executable, or assembler source file.

To choose the name for an object file, use this option in combination with the **-c** option. For an assembler source file, use it in combination with the **-S** option.

### **Defaults**

The default name for an executable file is **a.out**. The default name for an object or assembler source file is the same as the source file except that it has a **.o** or **.s** extension.

### **Rules**

Except when you specify the **-c** or **-S** option, the **-o** option is passed directly to the **ld** command, instead of being processed by XL Fortran.

### **Examples**

```
xlf95 t.f # Produces "a.out"
xlf95 -c t.f # Produces "t.o"
xlf95 -o test_program t.f # Produces "test_program"
xlf95 -S -o t2.s t.f # Produces "t2.s"
```

## -p option

### Syntax

-p[g]

Sets up the object file for profiling.

**-p** prepares the program for profiling. When you execute the program, it produces a **mon.out** file with the profiling information. You can then use the **prof** command to generate a run-time profile.

**-pg** is like **-p**, but it produces more extensive statistics. Running a program compiled with **-pg** produces a **gmon.out** file, which you use with the **gprof** command to generate a run-time profile.

### Rules

For profiling, the compiler produces monitoring code that counts the number of times each routine is called. The compiler replaces the startup routine of each subprogram with one that calls the monitor subroutine at the start. When the program ends normally, it writes the recorded information to the **mon.out** or **gmon.out** file.

### Examples

```
$ xlf95 -p needs_tuning.f
$ a.out
$ prof
.
```

profiling data

```
.
```

```
$ xlf95 -pg needs_tuning.f
$ a.out
$ gprof
```

```
.
```

detailed and verbose profiling data

```
.
```

### Related information

For more information on profiling and the **prof** and **gprof** commands, see the *AIX Commands Reference*.

## -Q option

### Syntax

`-Q+names` | `-Q-names` | `-Q` | `-Q!`

Specifies whether Fortran 90 or Fortran 95 procedures are inlined and/or the names of particular procedures that should or should not be inlined. *names* is a list of procedure names that are separated by colons.

### Rules

By default, `-Q` only affects internal or module procedures. To turn on inline expansion for calls to procedures in different scopes, you must also use the `-qipa` option.

### Arguments

The `-Q` option without any list inlines all appropriate procedures, subject to limits on the number of inlined calls and the amount of code size increase as a result. `+names` specifies the names, separated by colons, of procedures to inline and raises these limits for those procedures. `-names` specifies the names, separated by colons, of procedures not to inline. You can specify more than one of these options to precisely control which procedures are most likely to be inlined.

The `-Q!` option turns off inlining.

A procedure is not inlined by the basic `-Q` option unless it is quite small. In general, this means that it contains no more than several source statements (although the exact cutoff is difficult to determine). A procedure named by `-Q+` can be up to approximately 20 times larger and still be inlined.

### Restrictions

You must specify at least an optimization level of `-O2` for inlining to take effect with `-Q`.

If you specify inlining for a procedure, the following `@PROCESS` compiler directives are only effective if they come before the first compilation unit in the file: `ALIAS`, `ALIGN`, `ATTR`, `COMPACT`, `DBG`, `EXTCHK`, `EXTNAME`, `FLOAT`, `FLTTRAP`, `HALT`, `IEEE`, `LIST`, `MAXMEM`, `OBJECT`, `OPTIMIZE`, `PHSINFO`, `SPILLSIZE`, `STRICT`, and `XREF`.

### Examples

```
xlf95 -O -Q many_small_subprogs.f # Compiler decides what to inline.
xlf95 -O -Q+bigfunc:hugefunc test.f # Inline even though these are big.
xlf95 -O -Q -Q-only_once pi.f # Inline except for this one procedure.
```

### Related information

- “`-qipa` option” on page 169
- *Benefits of Interprocedural Analysis* in the *XL Fortran Optimization and Programming Guide*

## -q32 option

### Syntax

-q32

Enables 32-bit compilation mode (or, more briefly, 32-bit mode) support in a 64-bit environment. The **-q32** option indicates the compilation bit mode and, together with the **-qarch** option, determines the target machines that the 32-bit executable will run on.

### Rules

- The default integer and default real size are 4 bytes in 32-bit mode.
- The default integer pointer size is 4 bytes in 32-bit mode.
- 32-bit object modules are created when targeting 32-bit mode.
- **-q32** is the default, if you have not specified either **-q32** or **-q64** and if you have not set the **OBJECT\_MODE** environment variable. For a description of the **OBJECT\_MODE** environment variable, see “Default bit mode” on page 272.
- **-q64** can override **-q32**.
- All settings for **-qarch** are compatible with **-q32**. If you specify **-q32**, the default **-qarch** suboption is **com**, and the default **-qtune** suboption for **-q32** is **pwr2**.
- The **LOC** intrinsic returns an **INTEGER(4)** value.

### Examples

- Using 32-bit compilation mode and targeting a generic PowerPC architecture:

```
-qarch=ppc -q32
```

- Now keep the same compilation mode, but change the target to RS64II:

```
-qarch=ppc -q32 -qarch=rs64b
```

Notice that the last setting for **-qarch** wins.

- Now keep the same target, but change the compilation mode to 64-bit:

```
-qarch=ppc -q32 -qarch=rs64b -q64
```

Notice that specifying **-q64** overrides the earlier instance of **-q32**.

### Related information

- “-qarch option” on page 114
- “-qtune option” on page 240
- “-qwarn64 option” on page 248
- Chapter 6, “Using XL Fortran in a 64-Bit Environment,” on page 271
- “-q64 option” on page 106

## -q64 option

### Syntax

`-q64[=targettype]`

Indicates the 64-bit compilation bit mode and, together with the `-qarch` option, determines the target machines on which the 64-bit executable will run. The `-q64` option indicates that the object module will be created in 64-bit object format and that the 64-bit instruction set will be generated. Note that you may compile in a 32-bit environment to create 64-bit objects, but you must link them in a 64-bit environment with the `-q64` option.

### Defaults

`-q64=targettype` is always true when compiling with `-q64` on AIX 5.1 and above. The `-q64=targettype` suboption is for compatibility with older XL Fortran compilers. It is not necessary to specify it to generate the 64-bit LDT ABI. `-q64=nolargetype` is no longer a supported compiler option and will generate a warning if used.

### Rules

- Settings for `-qarch` that are compatible with `-q64` are as follows:
  - `-qarch=auto` (if compiling on a 64-bit system)
  - `-qarch=com` (With `-q64` and `-qarch=com`, the compiler will silently upgrade the arch setting to `ppc64`.)
  - `-qarch=ppc` (With `-q64` and `-qarch=ppc`, the compiler will silently upgrade the arch to `ppc64`.)
  - `-qarch=ppcgr` (With `-q64` and `-qarch=ppcgr`, the compiler will silently upgrade the arch to `ppc64gr`.)
  - `-qarch=ppc64`
  - `-qarch=ppc64v`
  - `-qarch=ppc64gr`
  - `-qarch=ppc64grsq`
  - `-qarch=rs64a`
  - `-qarch=rs64b`
  - `-qarch=rs64c`
  - `-qarch=pwr3`
  - `-qarch=pwr4`
  - `-qarch=pwr5`
  - `-qarch=pwr5x`
  - `-qarch=ppc970`
- The default `-qarch` setting for `-q64` is `ppc64`.
- 64-bit object modules are created when targeting 64-bit mode.
- `-q32` may override `-q64`.
- `-q64` will override a conflicting setting for `-qarch`.

For example,

```
-q64 -qarch=604
```

will be changed to the setting `-qarch=ppc64`.

- The default tune setting for `-q64` is `-qtune=pwr3`.
- The default integer and default real size is 4 bytes in 64-bit mode.
- The default integer pointer size is 8 bytes in 64-bit mode.
- The maximum array size increases to approximately  $2^{40}$  bytes (in static storage) or  $2^{60}$  bytes (in dynamic allocation on the heap). The maximum dimension bound range is extended to  $-2^{63}$ ,  $2^{63}-1$  bytes. The maximum array size for array constants has not been extended and will remain the same as the maximum in 32-bit mode. The maximum array size that you can initialize is  $2^{28}$  bytes.

- The maximum iteration count for array constructor implied DO loops increases to  $2^{63}-1$  bytes.
- The maximum character variable length extends to approximately  $2^{40}$  bytes. The maximum length of character constants and subobjects of constants remains the same as in 32-bit mode, which is 32 767 bytes (32 KB).
- The **LOC** intrinsic returns an **INTEGER(8)** value.
- If you must use **-qautodbl=dblpad** in 64-bit mode, you should use **-qintsize=8** to promote **INTEGER(4)** to **INTEGER(8)** for 8 byte integer arithmetic.

## Restrictions

- Objects that are generated with the 64-bit LDT ABI are not compatible with 64-bit non-LDT ABI objects. 64-bit LDT ABI objects cannot be linked with 64-bit non-LDT ABI objects on any level of AIX. XL Fortran no longer supports non-LDT ABI objects either for producing them or linking with them.
- 64-bit LDT ABI objects must be linked on AIX 5.1 and above. Objects created with the old 64-bit non-LDT ABI must be linked on AIX 4.3.3. XL Fortran no longer supports emitting code that executes on AIX version 4.
- 64-bit LDT ABI applications cannot be loaded or executed on AIX 4.3.3. 64-bit non-LDT ABI applications cannot be loaded or executed on AIX 5.1 and above. **Existing 64-bit applications compiled on AIX 4.3.3 must be recompiled to run on AIX 5.1 and above!**
- 64-bit module (.mod) files created by previous versions of XL Fortran can only be used if they were compiled with **-q64=largetype**.
- The compiler no longer appends **ldt** to the specified directory name at compilation time with **-q64=largetype** if the **include\_64** attribute is used in the **xlf.cfg** file to specify an alternate directory for 64-bit include and module files. As a result, you may need to change your existing build environment.
- Each of the following situations will produce an error message:
  1. Attempting to link, load, or execute conflicting 64-bit ABI objects on any level of AIX
  2. Attempting to link, load, or execute 64-bit LDT ABI objects on AIX 4.3.3
  3. Attempting to link, load, or execute 64-bit non-LDT ABI objects on AIX 5.1 and above.

## Examples

This example targets the RS64II (also known as RS64b) in 64-bit mode:

```
-q32 -qarch=rs64b -q64
```

In this example 64-bit compilation that targets the common group of 64-bit architectures (which currently consists only of the RS64I, RS64II, RS64III, POWER3, POWER4, POWER5, POWER5+, and PowerPC 970):

```
-q64 -qarch=com
```

arch setting is silently upgraded to **ppc64**, the most "common" 64-bit mode compilation target.

In this example, the **-qarch** option conflicts with **-q64**:

```
-qarch=604 -q64
```

which results in a suboption setting of **-q64 -qarch=ppc64** and a warning message.

In the example that follows, the **-qarch** option conflicts with **-q64**:

```
-q64 -qarch=604
```

which results in a suboption setting of **-q64 -qarch=ppc64** and a warning message.

**Related information**

- “-qarch option” on page 114
- “-qtune option” on page 240
- Chapter 6, “Using XL Fortran in a 64-Bit Environment,” on page 271
- “-qwarn64 option” on page 248

## -qalias option

### Syntax

```
-qalias={argument_list}
ALIAS({ARGUMENT_LIST})
```

Indicates whether a program contains certain categories of aliasing. The compiler limits the scope of some optimizations when there is a possibility that different names are aliases for the same storage locations. See *Optimizing XL compiler programs* in the *XL Fortran Optimization and Programming Guide* for information on aliasing strategies you should consider.

### Arguments

#### aryovrlp | **noaryovrlp**

Indicates whether the compilation units contain any array assignments between storage-associated arrays. If not, specify **noaryovrlp** to improve performance.

#### intptr | **nointptr**

Indicates whether the compilation units contain any integer **POINTER** statements. If so, specify **INTPTR**.

#### pteovrlp | **nopteovrlp**

Indicates whether any pointee variables may be used to refer to any data objects that are not pointee variables, or whether two pointee variables may be used to refer to the same storage location. If not, specify **NOPTEOVRLP**.

#### std | **nostd**

Indicates whether the compilation units contain any nonstandard aliasing (which is explained below). If so, specify **nostd**.

### Rules

An alias exists when an item in storage can be referred to by more than one name. The Fortran 90 and Fortran 95 standards allow some types of aliasing and disallow some others. The sophisticated optimizations that the XL Fortran compiler performs increase the likelihood of undesirable results when nonstandard aliasing is present, as in the following situations:

- The same data object is passed as an actual argument two or more times in the same subprogram reference. The aliasing is not valid if either of the actual arguments becomes defined, undefined, or redefined.
- A subprogram reference associates a dummy argument with an object that is accessible inside the referenced subprogram. The aliasing is not valid if any part of the object associated with the dummy argument becomes defined, undefined, or redefined other than through a reference to the dummy argument.
- A dummy argument becomes defined, undefined, or redefined inside a called subprogram in some other way than through the dummy argument.
- Subscripting beyond the bounds of an array within a common block.

### Applicable product levels

**-qalias=nostd** replaces the option **-qxflag=xalias** and makes it obsolete.

The introduction of the **-qipa** option does not remove the need for **-qalias**.

## Examples

If the following subroutine is compiled with `-qalias=nopteovrlp`, the compiler may be able to generate more efficient code. You can compile this subroutine with `-qalias=nopteovrlp`, because the integer pointers, `ptr1` and `ptr2`, point at dynamically allocated memory only.

```
subroutine sub(arg)
 real arg
 pointer(ptr1, pte1)
 pointer(ptr2, pte2)
 real pte1, pte2

 ptr1 = malloc(%val(4))
 ptr2 = malloc(%val(4))
 pte1 = arg*arg
 pte2 = int(sqrt(arg))
 arg = pte1 + pte2
 call free(%val(ptr1))
 call free(%val(ptr2))
end subroutine
```

If most array assignments in a compilation unit involve arrays that do not overlap but a few assignments do involve storage-associated arrays, you can code the overlapping assignments with an extra step so that the **NOARYOVRLP** suboption is still safe to use.

```
@PROCESS ALIAS(NOARYOVRLP)
! The assertion that no array assignments involve overlapping
! arrays allows the assignment to be done without creating a
! temporary array.
program test
 real(8) a(100)
 integer :: j=1, k=50, m=51, n=100

 a(1:50) = 0.0d0
 a(51:100) = 1.0d0

 ! Timing loop to achieve accurate timing results
 do i = 1, 1000000
 a(j:k) = a(m:n) ! Here is the array assignment
 end do

 print *, a
end program
```

In Fortran, this aliasing is not permitted if **J** or **K** are updated, and, if it is left undetected, it can have unpredictable results.

! We cannot assert that this unit is free  
! of array-assignment aliasing because of the assignments below.

```
subroutine sub1
 integer a(10), b(10)
 equivalence (a, b(3))
 a = b ! a and b overlap.
 a = a(10:1:-1) ! The elements of a are reversed.
end subroutine
```

! When the overlapping assignment is recoded to explicitly use a  
! temporary array, the array-assignment aliasing is removed.  
! Although ALIAS(NOARYOVRLP) does not speed up this assignment,  
! subsequent assignments of non-overlapping arrays in this unit  
! are optimized.

```
@PROCESS ALIAS(NOARYOVRLP)
subroutine sub2
 integer a(10), b(10), t(10)
```

```

equivalence (a, b(3))
t = b; a = t
t = a(10:1:-1); a = t
end subroutine

```

When **SUB1** is called, an alias exists between **J** and **K**. **J** and **K** refer to the same item in storage.

```

CALL SUB1(I,I)
...
SUBROUTINE SUB1(J,K)

```

In the following example, the program might store 5 instead of 6 into **J** unless **-qalias=nostd** indicates that an alias might exist.

```

INTEGER BIG(1000)
INTEGER SMALL(10)
COMMON // BIG
EQUIVALENCE(BIG,SMALL)
...
BIG(500) = 5
SMALL (I) = 6 ! Where I has the value 500
J = BIG(500)

```

## Restrictions

Because this option inhibits some optimizations of some variables, using it can lower performance.

Programs that contain nonstandard or integer **POINTER** aliasing may produce incorrect results if you do not compile them with the correct **-qalias** settings. The **xlf90**, **xlf90\_r**, **xlf90\_r7**, **xlf95**, **xlf95\_r**, **xlf95\_r7**, **f90**, and **f95** commands assume that a program contains only standard aliasing (**-qalias=aryovrlp:pteovrlp:std:nointptr**), while the **xlf\_r**, **xlf\_r7**, **xlf**, and **f77/fort77** commands, for compatibility with XL Fortran Version 2, assume that integer **POINTERS** may be present (**-qalias=aryovrlp:pteovrlp:std:intptr**).

## -qalign option

### Syntax

```
-qalign={ [no]4k | struct={suboption} | bindc={suboption} }
ALIGN({ [NO]4K | STRUCT{(suboption)} | BINDC{(suboption)} })
```

Specifies the alignment of data objects in storage, which avoids performance problems with misaligned data. The **[no]4k**, **bindc**, and **struct** options can be specified and are not mutually exclusive. The **[no]4k** option is useful primarily in combination with logical volume I/O and disk striping.

### Defaults

The default setting is **-qalign=no4k:struct=natural:bindc=power**.

### Arguments

**[no]4k** Specifies whether to align large data objects on page (4 KB) boundaries, for improved performance with data-stripped I/O. Objects are affected depending on their representation within the object file. The affected objects are arrays and structures that are 4 KB or larger and are in static or bss storage and also CSECTs (typically **COMMON** blocks) that are 8 KB or larger. A large **COMMON** block, equivalence group containing arrays, or structure is aligned on a page boundary, so the alignment of the arrays depends on their position within the containing object. Inside a structure of non-sequence derived type, the compiler adds padding to align large arrays on page boundaries.

#### **bindc={suboption}**

Specifies that the alignment and padding for an XL Fortran derived type with the BIND(C) attribute is compatible with a C struct type that is compiled with the corresponding XL C alignment option. The compatible alignment options include:

| XL Fortran Option                     | Corresponding<br>XL C Option    |
|---------------------------------------|---------------------------------|
| <b>-qalign=bindc=bit_packed</b>       | <b>-qalign=bit_packed</b>       |
| <b>-qalign=bindc=full   power</b>     | <b>-qalign=full   power</b>     |
| <b>-qalign=bindc=mac68k   twobyte</b> | <b>-qalign=mac68k   twobyte</b> |
| <b>-qalign=bindc=natural</b>          | <b>-qalign=natural</b>          |
| <b>-qalign=bindc=packed</b>           | <b>-qalign=packed</b>           |

#### **struct={suboption}**

The **struct** option specifies how objects or arrays of a derived type declared using a record structure are stored, and whether or not padding is used between components. All program units must be compiled with the same settings of the **-qalign=struct** option. The three suboptions available are:

##### **packed**

If the **packed** suboption of the **struct** option is specified, objects of a derived type are stored with no padding between components, other than any padding represented by **%FILL** components. The storage format is the same as would result for a sequence structure whose derived type was declared using a standard derived type declaration.

##### **natural**

If the **natural** suboption of the **struct** option is specified, objects of a derived type are stored with sufficient padding such that

components will be stored on their natural alignment boundaries, unless storage association requires otherwise. The natural alignment boundaries for objects of a type that appears in the left-hand column of the following table is shown in terms of a multiple of some number of bytes in the corresponding entry in the right-hand column of the table.

| Type                                        | Natural Alignment (in multiples of bytes) |
|---------------------------------------------|-------------------------------------------|
| INTEGER(1), LOGICAL(1), BYTE, CHARACTER     | 1                                         |
| INTEGER(2), LOGICAL(2)                      | 2                                         |
| INTEGER(4), LOGICAL(4), REAL(4)             | 4                                         |
| INTEGER(8), LOGICAL(8), REAL(8), COMPLEX(4) | 8                                         |
| REAL(16), COMPLEX(8), COMPLEX(16)           | 16                                        |
| Derived                                     | Maximum alignment of its components       |

If the **natural** suboption of the **struct** option is specified, arrays of derived type are stored so that each component of each element is stored on its natural alignment boundary, unless storage association requires otherwise.

### port

If the **port** suboption of the **struct** option is specified,

- Storage padding is the same as described above for the **natural** suboption, with the exception that the alignment of components of type complex is the same as the alignment of components of type real of the same kind.
- The padding for an object that is immediately followed by a union is inserted at the beginning of the first map component for each map in that union.

### Restrictions

The **port** suboption does not affect any arrays or structures with the **AUTOMATIC** attribute or arrays that are allocated dynamically. Because this option may change the layout of non-sequence derived types, when compiling programs that read or write such objects with unformatted files, use the same setting for this option for all source files.

You must use **-qalign=4k** if you are using the I/O techniques that are described in *Increasing throughput with logical volume I/O and data striping* of the *XL Fortran Optimization and Programming Guide*.

### Related information

You can tell if an array has the **AUTOMATIC** attribute and is thus unaffected by **-qalign=4k** if you look for the keywords **AUTOMATIC** or **CONTROLLED AUTOMATIC** in the listing of the “-qattr option” on page 120. This listing also shows the offsets of data objects.

## -qarch option

### Syntax

`-qarch=architecture`

Controls which instructions the compiler can generate. Changing the default can improve performance but might produce code that can only be run on specific machines.

In general, the **-qarch** option allows you to target a specific architecture for the compilation. For any given **-qarch** setting, the compiler defaults to a specific, matching **-qtune** setting, which can provide additional performance improvements. The resulting code may not run on other architectures, but it will provide the best performance for the selected architecture. To generate code that can run on more than one architecture, specify a **-qarch** suboption that supports a group of architectures, such as **com**, **ppc**, or **ppc64**; doing this will generate code that runs on all supported architectures, all PowerPC architectures, or all 64-bit PowerPC architectures, respectively. When a **-qarch** suboption is specified with a group argument, you can specify **-qtune** as either **auto**, or provide a specific architecture in the group. In the case of **-qtune=auto**, the compiler will generate code that runs on all architectures in the group specified by the **-qarch** suboption, but select instruction sequences that have best performance on the architecture of the machine used to compile. Alternatively you can target a specific architecture for tuning performance.

### Arguments

The choices for *architecture* are:

**auto** Automatically detects the specific architecture of the compiling machine. It assumes that the execution environment will be the same as the compilation environment. This option is implied if the **-O4** or **-O5** option is set or implied.

**com** You can run the executable file that the compiler generated on any hardware platform supported by the compiler, because the file contains only instructions that are common to all machines. This choice is the default if you specify **-q32**.

If you specify the **-q64** and **-qarch=com** options together, the target platform is 64-bit, and the **-qarch** option is silently upgraded to **ppc64**. The instruction set will be restricted to those instructions common to all 64-bit machines. See Chapter 6, “Using XL Fortran in a 64-Bit Environment,” on page 271 for details. Also, the **rndsngl** suboption of the **-qfloat** option is automatically turned on and cannot be turned off. While this yields better performance on PowerPC systems, you may get slightly different results than if you compile with **-qarch=com** and **-q32**.

**pwr** You can run the executable file on any POWER or POWER2 hardware platform. Because executable files for these platforms may contain instructions that are not available on PowerPC systems, they may be incompatible with those newer systems, or they may run more slowly because missing instructions are emulated through software traps.

**pwr2** You can run the executable file on any POWER2 hardware platform. Because executable files for these platforms may contain instructions that are not available on POWER and PowerPC (including POWER3) systems, they may be incompatible with those systems.

Note that **pwrx** is a synonym for **pwr2**, but **pwr2** is preferable.

- pwr2s** You can run the executable file on any desktop implementation of the POWER2 Chip. This architecture belongs to the **-qarch=pwr2** group.
- p2sc** You can run the executable file on any POWER2 Super Chip hardware platform. The POWER2 Super Chip belongs to the **-qarch=pwr2** group.
- 601** You can run the executable file on any PowerPC 601<sup>®</sup> hardware platform. Because the PowerPC 601 processor implements some instructions that are not present in other PowerPC implementations, programs might not run on other PowerPC processors. The **rndsngl** suboption of the **-qfloat** option is automatically turned on and cannot be turned off.
- 603** You can run the executable file on any PowerPC 603<sup>®</sup> hardware platform. Because the PowerPC 603 processor implements some instructions that are not present in other PowerPC implementations, such as the optional PowerPC graphics instructions, programs might not run on other PowerPC processors. The **rndsngl** suboption of the **-qfloat** option is automatically turned on and cannot be turned off.
- 604** You can run the executable file on any PowerPC 604<sup>®</sup> hardware platform. Because the PowerPC 604 processor implements some instructions that are not present in other PowerPC implementations, such as the optional PowerPC graphics instructions, programs might not run on other PowerPC processors. The **rndsngl** suboption of the **-qfloat** option is automatically turned on and cannot be turned off.
- ppc** You can run the executable file on any PowerPC hardware platform, including those that are based on the RS64I, RS64II, RS64III, 601, 603, 604, POWER3, POWER4, POWER5, PowerPC 970, and future PowerPC chips. If you specify the compiler option **-q64**, the target platform is 64-bit PowerPC, and the compiler silently upgrades the **-qarch** setting to **ppc64**. See Chapter 6, “Using XL Fortran in a 64-Bit Environment,” on page 271 for details.
- The **rndsngl** suboption of the **-qfloat** option is automatically turned on and cannot be turned off.
- ppcgr** In 32-bit mode, produces object code that may contain optional graphics instructions for PowerPC hardware platforms.
- In 64-bit mode, produces object code containing optional graphics instructions that will run on 64-bit PowerPC platforms, but not on 32-bit-only platforms, and the **-qarch** option will be silently upgraded to **-qarch=ppc64gr**.
- ppc64** You can run the executable file on any 64-bit PowerPC hardware platform. This suboption can be selected when compiling in 32-bit mode, but the resulting object code may include instructions that are not recognized or behave differently when run on PowerPC platforms that do not support 64-bit mode.
- ppc64gr** You can run the executable file on any 64-bit PowerPC hardware platform that supports the optional graphics instructions.
- ppc64grsq** You can run the executable file on any 64-bit PowerPC hardware platform that supports the optional graphics and square root instructions.
- rs64a** You can run the executable file on any RS64I machine.

- rs64b** You can run the executable file on any RS64II machine.
- rs64c** You can run the executable file on any RS64III machine.
- pwr3** You can run the executable file on any POWER3, POWER4, POWER5, POWER5+, or PowerPC 970 hardware platform. In previous releases, the **pwr3** setting was used to target the POWER3 and POWER4 group of processors. To have your compilation target a more general processor group, use the **ppc64grsq** setting, which includes the POWER3, POWER4, POWER5, POWER5+ or PowerPC 970 group of processors. Because executable files for these platforms may contain instructions that are not available on POWER, POWER2, or other PowerPC systems, they may be incompatible with those systems.
- pwr4** You can run the executable file on any POWER4, POWER5, POWER5+ or PowerPC 970 hardware platform. Use of **-qarch=pwr4** will result in binaries that will not run on most previous PowerPC implementations.
- pwr5** You can run the executable file on any POWER5 or POWER5+ hardware platform.
- pwr5x** You can run the executable file on any POWER5+ hardware platform.
- ppc64v**  
You can run the executable file on any 64-bit PowerPC hardware platform that supports the optional VMX instructions, such as a PowerPC 970.
- ppc970**  
You can run the executable file on any PowerPC 970 hardware platform.

**Notes:**

1. The **-qarch** setting determines the allowed choices and defaults for the **-qtune** setting. You can use **-qarch** and **-qtune** to target your program to particular machines.
2. Specifying **-q64** silently performs the following upgrades:
  - **ppc** to **ppc64**
  - **ppcgr** to **ppc64gr**
  - **com** to **ppc64**

If you intend your program to run only on a particular architecture, you can use the **-qarch** option to instruct the compiler to generate code specific to that architecture. This allows the compiler to take advantage of machine-specific instructions that can improve performance. The **-qarch** option provides arguments for you to specify certain chip models; for example, you can specify **-qarch=604** to indicate that your program will be executed on PowerPC 604 hardware platforms.

For a given application program, make sure that you specify the same **-qarch** setting when you compile each of its source files. Although the linker and loader may detect object files that are compiled with incompatible **-qarch** settings, you should not rely on it.

You can further enhance the performance of programs intended for specific machines by using other performance-related options like the **-qcache** and **-qhot** options.

Use these guidelines to help you decide whether to use this option:

- If your primary concern is to make a program widely distributable, keep the default (**com**). If your program is likely to be run on all types of processors

equally often, do not specify any **-qarch** or **-qtune** options. The default supports only the common subset of instructions of all processors.

- If you want your program to run on more than one architecture, but to be tuned to a particular architecture, use a combination of the **-qarch** and **-qtune** options. Make sure that your **-qarch** setting covers all the processor types you intend your program to run on. If you run such a program on an unsupported processor, your program may fail at execution time.
- If the program will only be used on a single machine or can be recompiled before being used on a different machine, specify the applicable **-qarch** setting. Doing so might improve performance and is unlikely to increase compile time. If you specify the **p2sc**, **pwr2**, **pwr2s**, **rs64a**, **rs64b**, **rs64c**, **601**, **603**, **604**, **pwr3**, **pwr4**, **pwr5**, **pwr5x** or **ppc970** suboption, you do not need to specify a separate **-qtune** option.
- If your primary concern is execution performance, you may see some speedup if you specify the appropriate **-qarch** suboption and perhaps also specify the **-qtune** and **-qcache** options. In this case, you may need to produce different versions of the executable file for different machines, which might complicate configuration management. You will need to test the performance gain to see if the additional effort is justified.
- It is usually better to target a specific architecture so your program can take advantage of the targeted machine's characteristics. For example, specifying **-qarch=pwr4** when targeting a POWER4 machine will benefit those programs that are floating-point intensive or have integer multiplies. On PowerPC systems, programs that process mainly unpromoted single-precision variables are more efficient when you specify **-qarch=ppc**. On POWER2 and POWER3 systems, programs that process mainly double-precision variables (or single-precision variables promoted to double by one of the **-qautodbl** options) become more efficient with **-qarch=pwr2**, **-qarch=pwr3**, **-qarch=pwr4** and **-qarch=pwr5**. The **-qautodbl=dblpad4** option will improve POWER and POWER2, but not POWER3, POWER4, and POWER5, which are PowerPC processors.

## Other considerations

The PowerPC instruction set includes two optional instruction groups that may be implemented by a particular hardware platform, but are not required. These two groups are the graphics instruction group and the sqrt instruction group. Code compiled with specific **-qarch** options (all of which refer to specific PowerPC machines) will run on any equivalent PowerPC machine that has an identical instruction group. The following table illustrates the instruction groups that are included for the various PowerPC machines.

Table 16. Instruction groups for PowerPC platforms

| Processor | Graphics group | sqrt group | 64-bit |
|-----------|----------------|------------|--------|
| 601       | no             | no         | no     |
| 603       | yes            | no         | no     |
| 604       | yes            | no         | no     |
| ppc       | no             | no         | no     |
| ppcgr     | yes            | no         | no     |
| ppc64     | no             | no         | yes    |
| ppc64v    | yes            | yes        | yes    |
| ppc64gr   | yes            | no         | yes    |
| ppc64grsq | yes            | yes        | yes    |

Table 16. Instruction groups for PowerPC platforms (continued)

| Processor | Graphics group | sqrt group | 64-bit |
|-----------|----------------|------------|--------|
| rs64a     | no             | no         | yes    |
| rs64b     | yes            | yes        | yes    |
| rs64c     | yes            | yes        | yes    |
| pwr3      | yes            | yes        | yes    |
| pwr4      | yes            | yes        | yes    |
| pwr5      | yes            | yes        | yes    |
| pwr5x     | yes            | yes        | yes    |
| ppc970    | yes            | yes        | yes    |

If you compile code using the **-qarch=pwr3** option, the code will run on an RS64B hardware platform but may not run on an RS64A platform because the instruction groups are not identical. Similarly, code compiled with the **-qarch=603** option will run on a POWER3 machine, but may not run on a RS64A machine.

### Related information

- “Compiling for specific architectures” on page 28
- “-qtune option” on page 240
- “-qcache option” on page 124
- Choosing the best -qarch suboption in the *XL Fortran Optimization and Programming Guide*

## -qassert option

### Syntax

-qassert={**deps** | nodeps | itercnt=*n*}  
ASSERT(**DEPS**) | NODEPS | ITERCNT(*N*)

Provides information about the characteristics of the files that can help to fine-tune optimizations.

### Arguments

**deps** | **nodeps**

Specifies whether or not any loop-carried dependencies exist.

**itercnt=*n***

Specifies a value for unknown loop iteration counts.

### Related information

See the following:

- *Benefits of High Order Transformation* in the *XL Fortran Optimization and Programming Guide* for background information and instructions on using these assertions.
- The **ASSERT** directive in the *XL Fortran Language Reference*.

## **-qattr option**

### **Syntax**

`-qattr[=full] | -qnoattr`  
`ATTR[(FULL)] | NOATTR`

Specifies whether to produce the attribute component of the attribute and cross-reference section of the listing.

### **Arguments**

If you specify only `-qattr`, only identifiers that are used are reported. If you specify `-qattr=full`, all identifiers, whether referenced or not, are reported.

If you specify `-qattr` after `-qattr=full`, the full attribute listing is still produced.

You can use the attribute listing to help debug problems caused by incorrectly specified attributes or as a reminder of the attributes of each object while writing new code.

### **Related information**

See “Options that control listings and messages” on page 62 and “Attribute and cross-reference section” on page 298.

## -qautodbl option

### Syntax

`-qautodbl=setting`  
`AUTODBL(setting)`

Provides an automatic means of converting single-precision floating-point calculations to double-precision and of converting double-precision calculations to extended-precision.

You might find this option helpful in porting code where storage relationships are significant and different from the XL Fortran defaults. For example, programs that are written for the IBM VS FORTRAN compiler may rely on that compiler's equivalent option.

### Rules

Although the POWER and POWER2 floating-point units perform **REAL(4)** calculations internally using fast **REAL(8)** arithmetic, it is often better to have these calculations done entirely using data entities that are **REAL(8)** or **DOUBLE PRECISION**. If the calculations are coded using **REAL** or **REAL(4)** data entities, the **REAL(4)-REAL(8)-REAL(4)** conversions take away the extra precision and range and also lessen performance, even though the intermediate calculations are done in IEEE double-precision.

### Arguments

The **-qautodbl** suboptions offer different strategies to preserve storage relationships between objects that are promoted or padded and those that are not.

The settings you can use are as follows:

|                    |                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b><u>none</u></b> | Does not promote or pad any objects that share storage. This setting is the default.                                                                                                                                                                                                                                                                                                                  |
| <b>dbl4</b>        | Promotes floating-point objects that are single-precision (4 bytes in size) or that are composed of such objects (for example, <b>COMPLEX</b> or array objects): <ul style="list-style-type: none"><li>• <b>REAL(4)</b> is promoted to <b>REAL(8)</b>.</li><li>• <b>COMPLEX(4)</b> is promoted to <b>COMPLEX(8)</b>.</li></ul> This suboption requires the <b>libxlfpm4.a</b> library during linking. |
| <b>dbl8</b>        | Promotes floating-point objects that are double-precision (8 bytes in size) or that are composed of such objects: <ul style="list-style-type: none"><li>• <b>REAL(8)</b> is promoted to <b>REAL(16)</b>.</li><li>• <b>COMPLEX(8)</b> is promoted to <b>COMPLEX(16)</b>.</li></ul> This suboption requires the <b>libxlfpm8.a</b> library during linking.                                              |
| <b>dbl</b>         | Combines the promotions that <b>dbl4</b> and <b>dbl8</b> perform.<br>This suboption requires the <b>libxlfpm4.a</b> and <b>libxlfpm8.a</b> libraries during linking.                                                                                                                                                                                                                                  |
| <b>dblpad4</b>     | Performs the same promotions as <b>dbl4</b> and pads objects of other types (except <b>CHARACTER</b> ) if they could possibly share storage with promoted objects.<br>This suboption requires the <b>libxlfpm4.a</b> and <b>libxlfpad.a</b> libraries during linking.                                                                                                                                 |

- dblpad8** Performs the same promotions as **dbl8** and pads objects of other types (except **CHARACTER**) if they could possibly share storage with promoted objects.
- This suboption requires the **libxlfpm8.a** and **libxlfpad.a** libraries during linking.
- dblpad** Combines the promotions done by **dbl4** and **dbl8** and pads objects of other types (except **CHARACTER**) if they could possibly share storage with promoted objects.
- This suboption requires the **libxlfpm4.a**, **libxlfpm8.a**, and **libxlfpad.a** libraries during linking.

## Rules

If the appropriate **-qautodbl** option is specified during linking, the program is automatically linked with the necessary extra libraries. Otherwise, you must link them in manually.

- When you have both **REAL(4)** and **REAL(8)** calculations in the same program and want to speed up the **REAL(4)** operations without slowing down the **REAL(8)** ones, use **dbl4**. If you need to maintain storage relationships for promoted objects, use **dblpad4**. If you have few or no **REAL(8)** calculations, you could also use **dblpad**.
- If you want maximum precision of all results, you can use **dbl** or **dblpad**. **dbl4**, **dblpad4**, **dbl8**, and **dblpad8** select a subset of real types that have their precision increased.

By using **dbl4** or **dblpad4**, you can increase the size of **REAL(4)** objects without turning **REAL(8)** objects into **REAL(16)**s. **REAL(16)** is less efficient in calculations than **REAL(8)** is.

The **-qautodbl** option handles calls to intrinsics with arguments that are promoted; when necessary, the correct higher-precision intrinsic function is substituted. For example, if single-precision items are being promoted, a call in your program to **SIN** automatically becomes a call to **DSIN**.

## Restrictions

- Because these extra conversions do not apply to the PowerPC floating-point unit, this option may not produce any speedup on PowerPC machines. However, programs that you compile with it still work and gain extra precision on all machines.
- Because character data is not promoted or padded, its relationship with storage-associated items that are promoted or padded may not be maintained.
- If the storage space for a pointee is acquired through the system routine **malloc**, the size specified to **malloc** should take into account the extra space needed to represent the pointee if it is promoted or padded.
- If an intrinsic function cannot be promoted because there is no higher-precision specific name, the original intrinsic function is used, and the compiler displays a warning message.
- You must compile every compilation unit in a program with the same **-qautodbl** setting. To detect inconsistent **-qautodbl** settings, use the **-qextchk** option when compiling the source file.

## Related information

For background information on promotion, padding, and storage/value relationships and for some source examples, see “Implementation details for `-qautodbl` promotion and padding” on page 305.

“`-qrealize` option” on page 210 describes another option that works like `-qautodbl`, but it only affects items that are of default kind type and does not do any padding. If you specify both the `-qrealize` and the `-qautodbl` options, only `-qautodbl` takes effect. Also, `-qautodbl` overrides the `-qdp` option.

“Linking 32-bit non-SMP object files using the `ld` command” on page 33 explains how to manually link additional libraries with object files that you compiled with `-qautodbl`.

## -qcache option

### Syntax

```
-qcache=
{
 assoc=number |
 auto |
 cost=cycles |
 level=level |
 line=bytes |
 size=Kbytes |
 type={C|c|D|d|I|i}
}[:...]
```

Specifies the cache configuration for a specific execution machine. The compiler uses this information to tune program performance, especially for loop operations that can be structured (or *blocked*) to process only the amount of data that can fit into the data cache.

If you know exactly what type of system a program is intended to be executed on and that system has its instruction or data cache configured differently from the default case (as governed by the **-qtune** setting), you can specify the exact characteristics of the cache to allow the compiler to compute more precisely the benefits of particular cache-related optimizations.

For the **-qcache** option to have any effect, you must include the **level** and **type** suboptions and specify at least level 2 of **-O**.

- If you know some but not all of the values, specify the ones you do know.
- If a system has more than one level of cache, use a separate **-qcache** option to describe each level. If you have limited time to spend experimenting with this option, it is more important to specify the characteristics of the data cache than of the instruction cache.
- If you are not sure of the exact cache sizes of the target systems, use relatively small estimated values. It is better to have some cache memory that is not used than to have cache misses or page faults from specifying a cache that is larger than the target system has.

### Arguments

**assoc=number**

Specifies the set associativity of the cache:

- 0** Direct-mapped cache
- 1** Fully associative cache
- n > 1** *n*-way set-associative cache

**auto** Automatically detects the specific cache configuration of the compiling machine. It assumes that the execution environment will be the same as the compilation environment.

**cost=cycles**

Specifies the performance penalty that results from a cache miss so that the compiler can decide whether to perform an optimization that might result in extra cache misses.

**level=level**

Specifies which level of cache is affected:

- 1** Basic cache

- 2 Level-2 cache or the table lookaside buffer (TLB) if the machine has no level-2 cache
- 3 TLB in a machine that does have a level-2 cache

Other levels are possible but are currently undefined. If a system has more than one level of cache, use a separate **-qcache** option to describe each level.

**line=bytes**

Specifies the line size of the cache.

**size=Kbytes**

Specifies the total size of this cache.

**type={C|c|D|d|I|i}**

Specifies the type of cache that the settings apply to, as follows:

- **C** or **c** for a combined data and instruction cache
- **D** or **d** for the data cache
- **I** or **i** for the instruction cache

## Restrictions

If you specify the wrong values for the cache configuration or run the program on a machine with a different configuration, the program may not be as fast as possible but will still work correctly. Remember, if you are not sure of the exact values for cache sizes, use a conservative estimate.

Currently, the **-qcache** option only has an effect when you also specify the **-qhot** option.

## Examples

To tune performance for a system with a combined instruction and data level-1 cache where the cache is two-way associative, 8 KB in size, and has 64-byte cache lines:

```
xlf95 -03 -qhot -qcache=type=c:level=1:size=8:line=64:assoc=2 file.f
```

To tune performance for a system with two levels of data cache, use two **-qcache** options:

```
xlf95 -03 -qhot -qcache=type=D:level=1:size=256:line=256:assoc=4 \
-qcache=type=D:level=2:size=512:line=256:assoc=2 file.f
```

To tune performance for a system with two types of cache, again use two **-qcache** options:

```
xlf95 -03 -qhot -qcache=type=D:level=1:size=256:line=256:assoc=4 \
-qcache=type=I:level=1:size=512:line=256:assoc=2 file.f
```

## Related information

See “-qtune option” on page 240, “-qarch option” on page 114, and “-qhot option” on page 159.

## **-qcclines option**

### **Syntax**

`-qcclines` | `-qnocclines`  
`CCLINES` | `NOCCLINES`

Determines whether the compiler recognizes conditional compilation lines in fixed source form and F90 free source form. IBM free source form is not supported.

### **Defaults**

The default is `-qcclines` if the `-qsmp=omp` option is turned on; otherwise, the default is `-qnocclines`.

### **Related information**

See *Conditional Compilation* in the *Language Elements* section of the *XL Fortran Language Reference*.

## **-qcheck option**

### **Syntax**

`-qcheck` | `-qnocheck`  
`CHECK` | `NOCHECK`

`-qcheck` is the long form of the “-C option” on page 89.

## -qci option

### Syntax

`-qci=numbers`  
`CI(numbers)`

Specifies the identification numbers (from 1 to 255) of the **INCLUDE** lines to process. If an **INCLUDE** line has a number at the end, the file is only included if you specify that number in a **-qci** option. The set of identification numbers that is recognized is the union of all identification numbers that are specified on all occurrences of the **-qci** option.

This option allows a kind of conditional compilation because you can put code that is only sometimes needed (such as debugging **WRITE** statements, additional error-checking code, or XLF-specific code) into separate files and decide for each compilation whether to process them.

### Examples

```
REAL X /1.0/
INCLUDE 'print_all_variables.f' 1
X = 2.5
INCLUDE 'print_all_variables.f' 1
INCLUDE 'test_value_of_x.f' 2
END
```

In this example, compiling without the **-qci** option simply declares **X** and assigns it a value. Compiling with **-qci=1** includes two instances of an include file, and compiling with **-qci=1:2** includes both include files.

### Restrictions

Because the optional number in **INCLUDE** lines is not a widespread Fortran feature, using it may restrict the portability of a program.

### Related information

See the section on the **INCLUDE** directive in the *XL Fortran Language Reference*.

## **-qcompact option**

### **Syntax**

`-qcompact` | `-qnocompact`  
`COMPACT` | `NOCOMPACT`

Reduces optimizations that increase code size.

By default, some techniques the optimizer uses to improve performance, such as loop unrolling and array vectorization, may also make the program larger. For systems with limited storage, you can use **-qcompact** to reduce the expansion that takes place. If your program has many loop and array language constructs, using the **-qcompact** option will affect your application's overall performance. You may want to restrict using this option to those parts of your program where optimization gains will remain unaffected.

### **Rules**

With **-qcompact** in effect, **-Q** and other optimization options still work; the reductions in code size come from limiting code replication that is done automatically during optimization.

## **-qcr option**

### **Syntax**

**-qcr** | **-qnocr**

Allows you to control how the compiler interprets the CR (carriage return) character. By default, the CR (Hex value X'0d') or LF (Hex value X'0a') character, or the CRLF (Hex value X'0d0a') combination indicates line termination in a source file. This allows you to compile code written using a Mac OS or DOS/Windows editor.

If you specify **-qnocr**, the compiler recognizes only the LF character as a line terminator. You must specify **-qocr** if you use the CR character for a purpose other than line termination.

## -qctyp1ss option

### Syntax

-qctyp1ss[(=[no]arg)] | -qnoctyp1ss  
CTYPLSS[(=[NO]ARG)] | NOCTYPLSS

Specifies whether character constant expressions are allowed wherever typeless constants may be used. This language extension might be needed when you are porting programs from other platforms.

### Arguments

**arg** | noarg      Suboptions retain the behavior of **-qctyp1ss**. Additionally, **arg** specifies that Hollerith constants used as actual arguments will be treated as integer actual arguments.

### Rules

With **-qctyp1ss**, character constant expressions are treated as if they were Hollerith constants and thus can be used in logical and arithmetic expressions.

### Restrictions

- If you specify the **-qctyp1ss** option and use a character-constant expression with the **%VAL** argument-list keyword, a distinction is made between Hollerith constants and character constants: character constants are placed in the rightmost byte of the register and padded on the left with zeros, while Hollerith constants are placed in the leftmost byte and padded on the right with blanks. All of the other **%VAL** rules apply.
- The option does not apply to character expressions that involve a constant array or subobject of a constant array at any point.

### Examples

**Example 1:** In the following example, the compiler option **-qctyp1ss** allows the use of a character constant expression.

```
@PROCESS CTYPLSS
 INTEGER I,J
 INTEGER, PARAMETER :: K(1) = (/97/)
 CHARACTER, PARAMETER :: C(1) = (/A'/)

 I = 4HABCD ! Hollerith constant
 J = 'ABCD' ! I and J have the same bit representation

! These calls are to routines in other languages.
 CALL SUB(%VAL('A')) ! Equivalent to CALL SUB(97)
 CALL SUB(%VAL(1HA)) ! Equivalent to CALL SUB(1627389952)

! These statements are not allowed because of the constant-array
! restriction.
! I = C // C
! I = C(1)
! I = CHAR(K(1))
END
```

**Example 2:** In the following example, the variable *J* is passed by reference. The suboption **arg** specifies that the Hollerith constant is passed as if it were an integer actual argument.

```
@PROCESS CTYPLSS(ARG)
 INTEGER :: J

 J = 3HIBM
! These calls are to routines in other languages.
 CALL SUB(J)
 CALL SUB(3HIBM) ! The Hollerith constant is passed as if
 ! it were an integer actual argument
```

### **Related information**

See *Hollerith Constants* in the *XL Fortran Language Reference* and *Passing arguments by reference or by value* in the *XL Fortran Optimization and Programming Guide*.

## **-qdbg option**

### **Syntax**

-qdbg | -qnodbg  
DBG | **NODBG**

**-qdbg** is the long form of the “-g option” on page 94.

## -qddim option

### Syntax

-qddim | **-qnoddim**  
DDIM | **NODDIM**

Specifies that the bounds of pointee arrays are re-evaluated each time the arrays are referenced and removes some restrictions on the bounds expressions for pointee arrays.

### Rules

By default, a pointee array can only have dimension declarators containing variable names if the array appears in a subprogram, and any variables in the dimension declarators must be dummy arguments, members of a common block, or use- or host-associated. The size of the dimension is evaluated on entry to the subprogram and remains constant during execution of the subprogram.

With the **-qddim** option:

- The bounds of a pointee array are re-evaluated each time the pointee is referenced. This process is called *dynamic dimensioning*. Because the variables in the declarators are evaluated each time the array is referenced, changing the values of the variables changes the size of the pointee array.
- The restriction on the variables that can appear in the array declarators is lifted, so ordinary local variables can be used in these expressions.
- Pointee arrays in the main program can also have variables in their array declarators.

### Examples

```
@PROCESS DDIM
INTEGER PTE, N, ARRAY(10)
POINTER (P, PTE(N))
DO I=1, 10
 ARRAY(I)=I
END DO
N = 5
P = LOC(ARRAY(2))
PRINT *, PTE ! Print elements 2 through 6.
N = 7 ! Increase the size.
PRINT *, PTE ! Print elements 2 through 8.
END
```

## -qdirective option

### Syntax

`-qdirective[=directive_list] | -qnodirective[=directive_list]`  
`DIRECTIVE[(directive_list)] | NODIRECTIVE[(directive_list)]`

Specifies sequences of characters, known as trigger constants, that identify comment lines as compiler comment directives.

### Background information

A compiler comment directive is a line that is not a Fortran statement but is recognized and acted on by the compiler. To allow you maximum flexibility, any new directives that might be provided with the XL Fortran compiler in the future will be placed inside comment lines. This avoids portability problems if other compilers do not recognize the directives.

### Defaults

The compiler recognizes the default trigger constant **IBM\***. Specification of **-qsmp** implies **-qdirective=smp\\$:\*\$omp*:ibmp**, and, by default, the trigger constants **SMP\$**, **\$OMP**, and **IBMP** are also turned on. If you specify **-qsmp=omp**, the compiler ignores all trigger constants that you have specified up to that point and recognizes only the **\$OMP** trigger constant. Specification of **-qthreaded** implies **-qdirective=ibmt**, and, by default, the trigger constant **IBMT** is also turned on.

### Arguments

The **-qnodirective** option with no *directive\_list* turns off all previously specified directive identifiers; with a *directive\_list*, it turns off only the selected identifiers.

**-qdirective** with no *directive\_list* turns on the default trigger constant **IBM\*** if it has been turned off by a previous **-qnodirective**.

### Notes

- Multiple **-qdirective** and **-qnodirective** options are additive; that is, you can turn directive identifiers on and off again multiple times.
- One or more *directive\_lists* can be applied to a particular file or compilation unit; any comment line beginning with one of the strings in the *directive\_list* is then considered to be a compiler comment directive.
- The trigger constants are not case-sensitive.
- The characters ( , ) , ' , " , ; , = , comma, and blank cannot be part of a trigger constant.
- To avoid wildcard expansion in trigger constants that you might use with these options, you can enclose them in single quotation marks on the command line. For example:  

```
xlf95 -qdirective='dbg*' -qnodirective='IBM*' directives.f
```
- This option only affects Fortran directives that the XL Fortran compiler provides, not those that any preprocessors provide.

## Examples

```
! This program is written in Fortran free source form.
PROGRAM DIRECTV
INTEGER A, B, C, D, E, F
A = 1 ! Begin in free source form.
B = 2
!OLDSTYLE SOURCEFORM(FIXED)
! Switch to fixed source form for this include file.
 INCLUDE 'set_c_and_d.inc'
!IBM* SOURCEFORM(FREE)
! Switch back to free source form.
E = 5
F = 6
END
```

For this example, compile with the option **-qdirective=oldstyle** to ensure that the compiler recognizes the **SOURCEFORM** directive before the **INCLUDE** line. After processing the include-file line, the program reverts back to free source form, after the **SOURCEFORM(FREE)** statement.

## Related information

See the section on the **SOURCEFORM** directive in the *XL Fortran Language Reference*.

As the use of incorrect trigger constants can generate warning messages or error messages or both, you should check the particular directive statement in the *Directives* section of the *XL Fortran Language Reference* for the suitable associated trigger constant.

## **-qdirectstorage option**

### **Syntax**

`-qdirectstorage` | `-qnodirectstorage`

Informs the compiler that a given compilation unit may reference write-through-enabled or cache-inhibited storage.

Use this option with discretion. It is intended for programmers who know how the memory and cache blocks work, and how to tune their applications for optimal performance. For a program to execute correctly on all PowerPC implementations of cache organization, the programmer should assume that separate instruction and data caches exist, and should program to the separate cache model.

**Note:** Using the `-qdirectstorage` option together with the `CACHE_ZERO` directive may cause your program to fail, or to produce incorrect results..

## **-qdlines option**

### **Syntax**

`-qdlines` | `-qnodlines`  
`DLINES` | `NODLINES`

`-qdlines` is the long form of the “-D option” on page 91.

## -qdp option

### Syntax

```
-qdp[=e] | -qnodpc
DPC[(E)] | NODPC
```

Increases the precision of real constants, for maximum accuracy when assigning real constants to **DOUBLE PRECISION** variables. This language extension might be needed when you are porting programs from other platforms.

### Rules

If you specify **-qdp**, all basic real constants (for example, 1.1) are treated as double-precision constants; the compiler preserves some digits of precision that would otherwise be lost during the assignment to the **DOUBLE PRECISION** variable. If you specify **-qdp=e**, all single-precision constants, including constants with an e exponent, are treated as double-precision constants.

This option does not affect constants with a kind type parameter specified.

### Examples

```
@process nodpc
 subroutine nodpc
 real x
 double precision y
 data x /1.000000000001/ ! The trailing digit is lost
 data y /1.000000000001/ ! The trailing digit is lost

 print *, x, y, x .eq. y ! So x is considered equal to y
 end

@process dpc
 subroutine dpc
 real x
 double precision y
 data x /1.000000000001/ ! The trailing digit is lost
 data y /1.000000000001/ ! The trailing digit is preserved

 print *, x, y, x .eq. y ! So x and y are considered different
 end

 program testdpc
 call nodpc
 call dpc
 end
```

When compiled, this program prints the following:

```
1.000000000 1.000000000000000000 T
1.000000000 1.0000000000100009 F
```

showing that with **-qdp** the extra precision is preserved.

### Related information

“-qautodbl option” on page 121 and “-qrealize option” on page 210 are more general-purpose options that can also do what **-qdp** does. **-qdp** has no effect if you specify either of these options.

## **-qdpcl option**

### **Syntax**

`-qdpcl` | `-qnodpc1`  
DPCL | `NODPCL`

Generates symbols that tools based on the Dynamic Probe Class Library (DPCL) can use to see the structure of an executable file.

When you specify the **-qdpcl** option, the compiler emits symbols to define blocks of code in a program. You can then use tools that use the DPCL interface to examine performance information, such as memory usage, for object files that you compiled with this option.

For more information on DPCL, visit the following URL:

<http://dpcl.sourceforge.net>

### **Restrictions**

You must also specify the **-g** option when you specify **-qdpcl**.

**-qdpcl** is not supported at **-O2** or higher optimization levels. If **-O2** is specified or implied by other options, **-qdpcl** will be disabled.

## **-qenablevmx option**

### **Syntax**

`-qenablevmx` | `-qnoenablevmx`

Enables the generation of VMX (Vector Multimedia eXtension) instructions.

`-qnoenablevmx` is set by default. To take advantage of architectures that support the VMX instruction set, you must explicitly specify the `-qenablevmx` option.

### **Notes**

- Some processors are able to support VMX (Vector Multimedia Extension) instructions. These instructions can offer higher performance when used with algorithmic-intensive tasks such as multimedia applications.
- The `-qenablevmx` compiler option enables generation of VMX instructions, but you should not use this option unless your operating system version supports the VMX instruction set.
- If `-qnoenablevmx` is in effect, `-qvecnvml` and `-qhot=simd` cannot be used.
- Specify `-qenablevmx` to use the VMX hardware intrinsics and `vector` data type. If you do not, the compiler will flag usage of the type and these intrinsics as an error.

### **Related information**

- “-qarch option” on page 114
- “-qhot option” on page 159
- “-qvecnvml option” on page 246
- VMX intrinsic procedures in the *XL Fortran Language Reference*
- Pixel, vector, and unsigned data types in the *XL Fortran Language Reference*

## -qenum option

### Syntax

-qenum=*value*

Specifies the range of the enumerator constant and enables storage size to be determined.

### Arguments

Regardless of its storage size, the enumerator's value will be limited by the range that corresponds to *value*. If the enumerator value exceeds the range specified, a warning message is issued and truncation is performed as necessary. The range limit and kind type parameter corresponding to each *value* is as follows:

Table 17. Enumerator sizes and types

| Value | Valid range of enumerator constant value    | Kind type parameter |
|-------|---------------------------------------------|---------------------|
| 1     | -128 to 127                                 | 4                   |
| 2     | -32768 to 32767                             | 4                   |
| 4     | -2147483648 to 2147483647                   | 4                   |
| 8     | -9223372036854775808 to 9223372036854775807 | 8                   |

### Related information

- ENUM / ENDENUM statement in the *XL Fortran Language Reference*

## -qescape option

### Syntax

```
-qescape | -qnoescape
ESCAPE | NOESCAPE
```

Specifies how the backslash is treated in character strings, Hollerith constants, H edit descriptors, and character string edit descriptors. It can be treated as an escape character or as a backslash character. This language extension might be needed when you are porting programs from other platforms.

### Defaults

By default, the backslash is interpreted as an escape character in these contexts. If you specify **-qnoescape**, the backslash is treated as the backslash character.

The default setting is useful for the following:

- Porting code from another Fortran compiler that uses the backslash as an escape character.
- Including “unusual” characters, such as tabs or newlines, in character data. Without this option, the alternative is to encode the ASCII values (or EBCDIC values, on mainframe systems) directly in the program, making it harder to port.

If you are writing or porting code that depends on backslash characters being passed through unchanged, specify **-qnoescape** so that they do not get any special interpretation. You could also write `\\` to mean a single backslash character under the default setting.

### Examples

```
$ # Demonstrate how backslashes can affect the output
$ cat escape.f
 PRINT *, 'a\bcde\fg'
 END
$ xlf95 escape.f
** _main === End of Compilation 1 ===
1501-510 Compilation successful for file escape.f.
$ a.out
cde
 g
$ xlf95 -qnoescape escape.f
** _main === End of Compilation 1 ===
1501-510 Compilation successful for file escape.f.
$ a.out
 a\bcde\fg
```

In the first compilation, with the default setting of **-qescape**, `\b` is printed as a backspace, and `\f` is printed as a formfeed character. With the **-qnoescape** option specified, the backslashes are printed like any other character.

### Related information

The list of escape sequences that XL Fortran recognizes is shown in *Escape sequences for character strings* *XL Fortran Optimization and Programming Guide*.

## **-qessl Option**

### **Syntax**

`-qessl` | `-qnoessl`

Allows the use of the Engineering and Scientific Subroutine Library (ESSL) routines in place of Fortran 90 intrinsic procedures.

The ESSL is a collection of subroutines that provides a wide range of mathematical functions for various scientific and engineering applications. The subroutines are tuned for performance on specific architectures. Some of the Fortran 90 intrinsic procedures have similar counterparts in ESSL. Performance is improved when these Fortran 90 intrinsic procedures are linked with ESSL. In this case, you can keep the interface of Fortran 90 intrinsic procedures, and get the added benefit of improved performance using ESSL.

### **Rules**

Use the ESSL Serial Library when linking with `-lessl`. Use the ESSL SMP Library when linking with `-lesslsmpl`.

Either `-lessl` or `-lesslsmpl` must be used whenever code is being compiled with `-qessl`. ESSL V4.1.1 or above is recommended. It supports both 32-bit and 64-bit environments.

The following MATMUL function calls may use ESSL routines when `-qessl` is enabled:

```
real a(10,10), b(10,10), c(10,10)
c=MATMUL(a,b)
```

### **Related information**

The ESSL libraries are not shipped with the XL Fortran compiler. For more information on these libraries, see the following URL:  
<http://publib.boulder.ibm.com/clresctr/windows/public/esslbooks.html>.

## **-qextchk option**

### **Syntax**

`-qextchk` | `-qnoextchk`  
`EXTCHK` | `NOEXTCHK`

Sets up type-checking information for common blocks, procedure definitions, procedure references, and module data. Later, the linker can detect mismatches across compilation units by using this information.

### **Rules**

At compile time, `-qextchk` verifies the consistency of procedure definitions and references and module data.

At link time, `-qextchk` verifies that actual arguments agree in type, shape, passing mode, and class with the corresponding dummy arguments and that declarations of common blocks and modules are consistent.

If null arguments are used in a procedure reference, the compiler will not verify that the actual arguments agree with the corresponding dummy arguments at both compile and link time.

## **-qextern option**

### **Syntax**

`-qextern=names`

Allows user-written procedures to be called instead of XL Fortran intrinsics. *names* is a list of procedure names separated by colons. The procedure names are treated as if they appear in an **EXTERNAL** statement in each compilation unit being compiled. If any of your procedure names conflict with XL Fortran intrinsic procedures, use this option to call the procedures in the source code instead of the intrinsic ones.

### **Arguments**

Separate the procedure names with colons.

### **Applicable product levels**

Because of the many Fortran 90 and Fortran 95 intrinsic functions and subroutines, you might need to use this option even if you did not need it for FORTRAN 77 programs.

### **Examples**

```
SUBROUTINE GETENV(VAR)
 CHARACTER(10) VAR
 PRINT *,VAR
END

CALL GETENV('USER')
END
```

Compiling this program with no options fails because the call to **GETENV** is actually calling the intrinsic subroutine, not the subroutine defined in the program. Compiling with **-qextern=getenv** allows the program to be compiled and run successfully.

## -qextname option

### Syntax

`-qextname[=name1[:name2...]]` | `-qnoextname`  
`EXTNAME[(name1: name2:...)]` | `NOEXTNAME`

Adds an underscore to the names of all global entities, which helps in porting programs from systems where this is a convention for mixed-language programs. Use `-qextname=name1[:name2...]` to identify a specific global entity (or entities). For a list of named entities, separate each name with a colon.

The name of a main program is not affected.

The `-qextname` option helps to port mixed-language programs to XL Fortran without modifications. Use of this option avoids naming problems that might otherwise be caused by:

- Fortran subroutines, functions, or common blocks that are named **main**, **MAIN**, or have the same name as a system subroutine
- Non-Fortran routines that are referenced from Fortran and contain an underscore at the end of the routine name

**Note:** XL Fortran Service and Utility Procedures, such as **flush\_** and **dtime\_**, have these underscores in their names already. By compiling with the `-qextname` option, you can code the names of these procedures without the trailing underscores.

- Non-Fortran routines that call Fortran procedures and use underscores at the end of the Fortran names
- Non-Fortran external or global data objects that contain an underscore at the end of the data name and are shared with a Fortran procedure

If your program has only a few instances of the naming problems that `-qextname` solves, you may prefer to select new names with the `-brename` option of the `ld` command.

### Restrictions

You must compile all the source files for a program, including the source files of any required module files, with the same `-qextname` setting.

If you use the **xlfortility** module to ensure that the Service and Utility subprograms are correctly declared, you must change the name to **xlfortility\_extname** when compiling with `-qextname`.

If there is more than one Service and Utility subprogram referenced in a compilation unit, using `-qextname` with no names specified and the **xlfortility\_extname** module may cause the procedure declaration check not to work accurately.

## Examples

```
@PROCESS EXTNAME
 SUBROUTINE STORE_DATA
 CALL FLUSH(10) ! Using EXTNAME, we can drop the final underscore.
 END SUBROUTINE

@PROCESS(EXTNAME(sub1))
program main
 external :: sub1, sub2
 call sub1() ! An underscore is added.
 call sub2() ! No underscore is added.
end program
```

## Related information

This option also affects the names that are specified in several other options, so you do not have to include underscores in their names on the command line. The affected options are “-qextern option” on page 146, “-Q option” on page 104, and “-qsigtrap option” on page 220.

## **-qfdpr option**

### **Syntax**

`-qfdpr` | `-qnofdpr`

Provides object files with information that the AIX Feedback Directed Program Restructuring (fdpr) performance-tuning utility needs to optimize the resulting executable file.

### **Restrictions**

The fdpr performance-tuning utility has its own set of restrictions, and it is not guaranteed to speed up all programs or produce executables that produce exactly the same results as the original programs.

If you use the **-qfdpr** compiler option, only those object files that are built with this flag will be reordered. Therefore, if you use **-qfdpr**, you should use it for all object files in a program. Static linking will not improve performance if you use the **-qfdpr** compiler option.

When you use **-qfdpr** on some of the objects that are built into an executable, fdpr will only perform some optimizations on the objects that are built with fdpr. This can mean that fdpr has less benefit on programs compiled using **-qfdpr**, because library code is not optimized (since it has not been compiled with **-qfdpr**).

The optimizations that the fdpr command performs are similar to those that the **-qpdf** option performs.

### **Related information**

For more information, see the fdpr man page and the *AIX Commands Reference*.

## **-qfixed option**

### **Syntax**

`-qfixed[=right_margin]`  
`FIXED[(right_margin)]`

Indicates that the input source program is in fixed source form and optionally specifies the maximum line length.

The source form specified when executing the compiler applies to all of the input files, although you can switch the form for a compilation unit by using a **FREE** or **FIXED @PROCESS** directive or switch the form for the rest of the file by using a **SOURCEFORM** comment directive (even inside a compilation unit).

For source code from some other systems, you may find you need to specify a right margin larger than the default. This option allows a maximum right margin of 132.

### **Defaults**

`-qfixed=72` is the default for the `xlf`, `xlf_r`, `xlf_r7`, `f77`, and `fort77` commands.

`-qfree=f90` is the default for the `f90`, `f95`, `xlf90`, `xlf90_r`, `xlf90_r7`, `xlf95`, `xlf95_r`, and `xlf95_r7` commands.

### **Related information**

See “-qfree option” on page 156.

For the precise specifications of this source form, see *Fixed Source Form* in the *XL Fortran Language Reference*.

## -qflag option

### Syntax

`-qflag=listing_severity:terminal_severity`  
`FLAG(listing_severity,terminal_severity)`

You must specify both *listing\_severity* and *terminal\_severity*.

Limits the diagnostic messages to those of a specified level or higher. Only messages with severity *listing\_severity* or higher are written to the listing file. Only messages with severity *terminal\_severity* or higher are written to the terminal. **-w** is a short form for **-qflag=e:e**.

### Arguments

The severity levels (from lowest to highest) are:

- i** Informational messages. They explain things that you should know, but they usually do not require any action on your part.
- l** Language-level messages, such as those produced under the **-qlanglvl** option. They indicate possible nonportable language constructs.
- w** Warning messages. They indicate error conditions that might require action on your part, but the program is still correct.
- e** Error messages. They indicate error conditions that require action on your part to make the program correct, but the resulting program can probably still be executed.
- s** Severe error messages. They indicate error conditions that require action on your part to make the program correct, and the resulting program will fail if it reaches the location of the error. You must change the **-qhalt** setting to make the compiler produce an object file when it encounters this kind of error.
- u** Unrecoverable error messages. They indicate error conditions that prevent the compiler from continuing. They require action on your part before you can compile your program.
- q** No messages. A severity level that can never be generated by any defined error condition. Specifying it prevents the compiler from displaying messages, even if it encounters unrecoverable errors.

The **-qflag** option overrides any **-qlanglvl** or **-qsaa** options specified.

### Defaults

The default for this option is **i:i**, which will show all compiler messages.

### Related information

See “-qlanglvl option” on page 177 and “Understanding XL Fortran error messages” on page 275.

## -qfloat option

### Syntax

`-qfloat=options`  
`FLOAT(options)`

Selects different strategies for speeding up or improving the accuracy of floating-point calculations.

This option replaces several separate options. For any new code, you should use it instead of **-qfold**, **-qmaf**, or related options.

You should be familiar with the information in *Implementation details of XL Fortran floating-point processing* in the *XL Fortran Optimization and Programming Guide* and the IEEE standard before attempting to change any **-qfloat** settings.

### Defaults

The default setting uses the suboptions **nofltint**, **fold**, **nohsflt**, **nohssngl**, **maf**, **nonans**, **norndsnl**, **norm**, **norsqrt**, and **nostrictnmaf**. Some options change this default, as explained below.

The default setting of each suboption remains in effect unless you explicitly change it. For example, if you select **-qfloat=nofold**, the settings for **nohsflt**, **nohssngl**, or related options are not affected.

### Arguments

The available suboptions each have a positive and negative form, such as **fold** and **nofold**, where the negative form is the opposite of the positive.

The suboptions are as follows:

#### fltint | **nofltint**

Speeds up floating-point-to-integer conversions by using an inline sequence of code instead of a call to a library function.

The library function, which is called by default if **-qfloat=fltint** is not specified or implied by another option, checks for floating-point values outside the representable range of integers and returns the minimum or maximum representable integer if passed an out-of-range floating-point value.

The Fortran language does not require checking for floating-point values outside the representable range of integers. In order to improve efficiency, the inline sequence used by **-qfloat=fltint** does not perform this check. If passed a value that is out of range, the inline sequence will produce undefined results.

Although this suboption is turned off by default, it is turned on by the **-O3** optimization level unless you also specify **-qstrict**.

#### fold | **nofold**

Evaluates constant floating-point expressions at compile time, which may yield slightly different results from evaluating them at run time. The compiler always evaluates constant expressions in specification statements, even if you specify **nofold**.

#### hsflt | **nohsflt**

Speeds up calculations by preventing rounding for single-precision expressions and by replacing floating-point division by multiplication with

the reciprocal of the divisor. It also uses the same technique as the **fltint** suboption for floating-point-to-integer conversions.

**Note:** Use **-qfloat=hsflt** on applications that perform complex division and floating-point conversions where floating-point calculations have known characteristics. In particular, all floating-point results must be within the defined range of representation of single precision. The use of this option when compiling other application programs may produce unexpected results without warning. Use with discretion. See “Technical details of the **-qfloat=hsflt** option” on page 304 for details.

**hssngl** | **nohssngl**

Speeds up calculations in a safer way than **hsflt**, by rounding single-precision expressions only when the results are stored into **REAL(4)** memory locations.

**maf** | **nomaf**

Makes floating-point calculations faster and more accurate by using multiply-add instructions where appropriate. The possible disadvantage is that results may not be exactly equivalent to those from similar calculations that are performed at compile time or on other types of computers. Also, negative zero may be produced.

**nans** | **nonans**

Allows you to use the **-qflttrap=invalid:enable** option to detect and deal with exception conditions that involve signaling NaN (not-a-number) values. Use this suboption only if your program explicitly creates signaling NaN values, because these values never result from other floating-point operations.

**rndsngl** | **norndsngl**

Rounds the result of each single-precision (**REAL(4)**) operation to single-precision, rather than waiting until the full expression is evaluated. It sacrifices speed for consistency with results from similar calculations on other types of computers.

This setting is always in effect for programs that you compile with any of the **-qarch** PowerPC suboptions, because of the way the PowerPC floating-point unit works. The **rndsngl** suboption is also turned on if you specify **-q64** and **-qarch=com** together.

**rrm** | **norrm**

Turns off compiler optimizations that require the rounding mode to be the default, round-to-nearest, at run time. Use this option if your program changes the rounding mode by any means, such as by calling the **fpsets** procedure. Otherwise, the program may compute incorrect results.

**rsqrt** | **norsqrt**

Speeds up some calculations by replacing division by the result of a square root with multiplication by the reciprocal of the square root.

Although this suboption is turned off by default, specifying **-O3** turns it on unless you also specify **-qstrict**.

**strictnmaf** | **nostrictnmaf**

Turns off floating-point transformations that are used to introduce negative MAF instructions, as these transformations do not preserve the sign of a zero value. By default, the compiler enables these types of transformations.

To ensure strict semantics, specify both **-qstrict** and **-qfloat=strictnmaf**.

## -qflttrap option

### Syntax

`-qflttrap[=suboptions] | -qnoflttrap`  
`FLTTRAP[(suboptions)] | NOFLTTRAP`

Determines what types of floating-point exception conditions to detect at run time. The program receives a **SIGTRAP** signal when the corresponding exception occurs.

### Arguments

|                   |                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ENable</b>     | Turn on checking for the specified exceptions in the main program so that the exceptions generate <b>SIGTRAP</b> signals. You must specify this suboption if you want to turn on exception trapping without modifying your source code.                                                                                                                                                    |
| <b>IMPrecise</b>  | Only check for the specified exceptions on subprogram entry and exit. This suboption improves performance, but it can make the exact spot of the exception difficult to find.                                                                                                                                                                                                              |
| <b>INEXact</b>    | Detect and trap on floating-point inexact if exception-checking is enabled. Because inexact results are very common in floating-point calculations, you usually should not need to turn this type of exception on.                                                                                                                                                                         |
| <b>INValid</b>    | Detect and trap on floating-point invalid operations if exception-checking is enabled.                                                                                                                                                                                                                                                                                                     |
| <b>NANQ</b>       | Detect and trap all quiet not-a-number values (NaNQs) and signaling not-a-number values (NaNSs). Trapping code is generated regardless of specifying the <b>enable</b> or <b>imprecise</b> suboption. This suboption detects all NaN values handled by or generated by floating point instructions, including those not created by invalid operations. This option can impact performance. |
| <b>OVerflow</b>   | Detect and trap on floating-point overflow if exception-checking is enabled.                                                                                                                                                                                                                                                                                                               |
| <b>UNDerflow</b>  | Detect and trap on floating-point underflow if exception-checking is enabled.                                                                                                                                                                                                                                                                                                              |
| <b>ZERODivide</b> | Detect and trap on floating-point division by zero if exception-checking is enabled.                                                                                                                                                                                                                                                                                                       |

### Defaults

The **-qflttrap** option without suboptions is equivalent to **-qflttrap=ov:und:zero:inv:inex**. However, because this default does not include **enable**, it is probably only useful if you already use **fpsets** or similar subroutines in your source. If you specify **-qflttrap** more than once, both with and without suboptions, the **-qflttrap** without suboptions is ignored.

The **-qflttrap** option is recognized during linking with IPA. Specifying the option at the link step overrides the compile-time setting.

### Restrictions

If you use **-qflttrap=inv:en** to compile a program containing an IEEE invalid **SQRT** operation without also specifying a **-qarch** target that implements the **sqrt** instruction set, the expected **SIGTRAP** signal will not occur when you run the program.

You can fix this problem for AIX Version 5.1 and higher by specifying the following command before running the program:

```
export SQRTE_EXCEPTION=3.1
```

## Examples

When you compile this program:

```
REAL X, Y, Z
DATA X /5.0/, Y /0.0/
Z = X / Y
PRINT *, Z
END
```

with the command:

```
xlf95 -qflttrap=zerodivide:enable -qsigtrap divide_by_zero.f
```

the program stops when the division is performed.

The **zerodivide** suboption identifies the type of exception to guard against. The **enable** suboption causes a **SIGTRAP** signal when the exception occurs. The **-qsigtrap** option produces informative output when the signal stops the program.

## Related information

- “-qsigtrap option” on page 220
- “-qarch option” on page 114
- *Detecting and trapping floating-point exceptions* in the *XL Fortran Optimization and Programming Guide* has full instructions on how and when to use the **-qflttrap** option, especially if you are just starting to use it.

## -qfree option

### Syntax

```
-qfree[={f90|ibm}]
FREE[({F90|IBM})]
```

Indicates that the source code is in free source form. The **ibm** and **f90** suboptions specify compatibility with the free source form defined for VS FORTRAN and Fortran 90, respectively. Note that the free source form defined for Fortran 90 also applies to Fortran 95.

The source form specified when executing the compiler applies to all of the input files, although you can switch the form for a compilation unit by using a **FREE** or **FIXED @PROCESS** directive or for the rest of the file by using a **SOURCEFORM** comment directive (even inside a compilation unit).

### Defaults

**-qfree** by itself specifies Fortran 90 free source form.

**-qfixed=72** is the default for the **xl f**, **xl f\_r**, **xl f\_r7**, **f77**, and **fort77** commands.

**-qfree=f90** is the default for the **f90**, **f95**, **xl f90**, **xl f90\_r**, **xl f90\_r7**, **xl f95**, **xl f95\_r**, and **xl f95\_r7** commands.

### Related information

See “-qfixed option” on page 150.

**-k** is equivalent to **-qfree=f90**.

Fortran 90 free source form is explained in *Free Source Form* in the *XL Fortran Language Reference*. It is the format to use for maximum portability across compilers that support Fortran 90 and Fortran 95 features now and in the future.

IBM free source form is equivalent to the free format of the IBM VS FORTRAN compiler, and it is intended to help port programs from the z/OS® platform. It is explained in *IBM Free Source Form* in the *XL Fortran Language Reference*.

## **-qfullpath option**

### **Syntax**

`-qfullpath` | `-qnofullpath`

Records the full, or absolute, path names of source and include files in object files compiled with debugging information. (Debug information is produced when you compile with the `-g` option.)

If you need to move an executable file into a different directory before debugging it or have multiple versions of the source files and want to ensure that the debugger uses the original source files, use the `-qfullpath` option in combination with the `-g` option so that source-level debuggers can locate the correct source files.

### **Defaults**

By default, the compiler records the relative path names of the original source file in each `.o` file. It may also record relative path names for include files.

### **Restrictions**

Although `-qfullpath` works without the `-g` option, you cannot do source-level debugging unless you also specify the `-g` option.

### **Examples**

In this example, the executable file is moved after being created, but the debugger can still locate the original source files:

```
$ xlf95 -g -qfullpath file1.f file2.f file3.f -o debug_version
...
$ mv debug_version $HOME/test_bucket
$ cd $HOME/test_bucket
$ dbx debug_version
```

### **Related information**

See “-g option” on page 94.

## -qhalt option

### Syntax

`-qhalt=severity`  
`HALT(severity)`

Stops before producing any object, executable, or assembler source files if the maximum severity of compile-time messages equals or exceeds the specified severity. *severity* is one of **i**, **l**, **w**, **e**, **s**, **u**, or **q**, meaning informational, language, warning, error, severe error, unrecoverable error, or a severity indicating “don’t stop”.

### Arguments

The severity levels (from lowest to highest) are:

- i** Informational messages. They explain things that you should know, but they usually do not require any action on your part.
- l** Language-level messages, such as those produced under the **-qlanglvl** option. They indicate possible nonportable language constructs.
- w** Warning messages. They indicate error conditions that might require action on your part, but the program is still correct.
- e** Error messages. They indicate error conditions that require action on your part to make the program correct, but the resulting program can probably still be executed.
- s** Severe error messages. They indicate error conditions that require action on your part to make the program correct, and the resulting program will fail if it reaches the location of the error. You must change the **-qhalt** setting to make the compiler produce an object file when it encounters this kind of error.
- u** Unrecoverable error messages. They indicate error conditions that prevent the compiler from continuing. They require action on your part before you can compile your program.
- q** No messages. A severity level that can never be generated by any defined error condition. Specifying it prevents the compiler from displaying messages, even if it encounters unrecoverable errors.

### Defaults

The default is **-qhalt=s**, which prevents the compiler from generating an object file when compilation fails.

### Restrictions

The **-qhalt** option can override the **-qobject** option, and **-qnoobject** can override **-qhalt**.

### Related information

- “-qhalt option”
- “-qobject option” on page 195

## -qhot option

### Syntax

`-qhot[=suboptions]` | `-qnohot`  
`HOT[=suboptions]` | `NOHOT`

Instructs the compiler to perform **high-order** loop analysis and **transformations** during optimization.

The **-qhot** compiler option is a powerful alternative to hand tuning that provides opportunities to optimize loops and array language. This compiler option will always attempt to optimize loops, regardless of the suboptions you specify.

If you do not specify an optimization level of **2** or higher when using **-O** and **-qhot**, the compiler implies **-O2**.

If you specify **-O3**, the compiler assumes **-qhot=level=0**. To prevent all HOT optimizations with **-O3**, you must specify **-qnohot**.

Specifying **-qhot** without suboptions implies **-qhot=nosimd**, **-qhot=noarraypad**, **-qhot=vector**, and **-qhot=level=1**. These suboptions are also implied by the options **-qsmp**, **-O4**, or **-O5**.

If you specify **-qhot**, **-qenablevmx**, and either **-qarch=ppc970** or **-qarch=ppc64v**, **-qhot=simd** will be set as the default.

**Array padding:** In XL Fortran, array dimensions that are powers of two can lead to a decrease in cache utilization. The **arraypad** suboption allows the compiler to increase the dimensions of arrays where doing so could improve the efficiency of array-processing loops. This can reduce cache misses and page faults that slow your array processing programs.

Specify **-qhot=arraypad** when your source includes large arrays with dimensions that are powers of 2. This can be particularly effective when the first dimension is a power of 2.

The **-C** option turns off some array optimizations.

**Vectorization:** The **-qhot** compiler option supports the **simd** and **vector** suboptions that can optimize loops in source code for operations on array data, by ensuring that operations run in parallel where applicable. Though both suboptions can provide an increase in performance, each suboption best applies to a particular type of vectorization.

*Short vectorization: -qhot=simd:* The **simd** suboption optimizes array data to run mathematical operations in parallel where the target architecture allows such operations. Parallel operations occur in 16-byte vector registers. The compiler divides vectors that exceed the register length into 16-byte units to facilitate optimization. A 16-byte unit can contain one of the following types of data:

- 4 4-byte units (for example, 4 integer(4)s, or 4 real(4)s)
- 8 2-byte units
- 16 1-byte units

Short vectorization does not support double-precision floating point mathematics. Users can typically see benefit when applying the **-qhot=simd** optimization to image processing applications, for example.

*Long vectorization: -qhot=vector:* The **vector** suboption, when used in conjunction with **-qnostrict** or an optimization level of **-O3** or higher, optimizes array data to run mathematical operations in parallel where applicable. The compiler uses standard registers with no vector size restrictions. Supporting single and double-precision floating-point mathematics, users can typically see benefit when applying **-qhot=vector** to applications with significant mathematical requirements.

## Arguments

### **arraypad**

The compiler pads any arrays where there could be an increase in cache utilization. Not all arrays will necessarily be padded, and the compiler can pad different arrays by different amounts.

### **arraypad=*n***

The compiler pads each array in the source. The pad amount must be a positive integer value. Each array will be padded by an integral number of elements. The integral value *n* must be multiples of the largest array element size for effective use of **arraypad**. This value is typically 4, 8, or 16.

When you specify the **arraypad** and **arraypad=*n*** options, the compiler does not check for reshaping or equivalences. If padding takes place, your program can produce unexpected results.

### **level={0 | 1}**

#### **level=0**

The compiler performs a subset of the of high-order transformations. Some of these include early distribution, loop interchange, and loop tiling, as examples.

Optimization level **-O3** implies **-qhot=level=0**.

#### **level=1**

**-qhot=level=1** is equivalent to **-qhot** and the compiler options that imply **-qhot** also imply **-qhot=level=1**, unless **-qhot=level=0** is explicitly specified.

The default hot level for all **-qhot** suboptions other than **level** is 1. For example specifying **-O3 -qhot=novector** sets the hot level to 1.

Specifying **-O3** implies **-qhot=level=0**, unless you explicitly specify **-qhot** or **-qhot=level=1**.

Any of **-O4**, **-O5**, or **-qsmp** implies **-qhot=level=1**, unless you explicitly specify **-qhot=level=0**.

### **simd | nosimd**

The compiler converts certain operations in a loop that apply to successive elements of an array into a call to a VMX (Vector Multimedia eXtension) instruction. This call calculates several results at one time, which is faster than calculating each result sequentially.

If you specify **-qhot=nosimd**, the compiler performs optimizations on loops and arrays, but avoids replacing certain code with calls to VMX instructions.

This suboption has effect only when the specified architecture supports VMX instructions and **-qenablevmx** is in effect.

**Note:** This suboption is supported on AIX 5L Version 5.3 with the 5300-03 Recommended Maintenance package and higher.

## **vector | novector**

When specified with **-qnostrict** or an optimization level of **-O3** or higher, the compiler converts certain operations (for example, square root, reciprocal square root) into a call to a MASS library routine that is in the **libxlopt.a** library. If the operations are in a loop, the vector version of the routine is called. If the operations are scalar, the scalar version of the routine is called. This call will calculate several results at one time, which is faster than calculating each result sequentially.

Since vectorization can affect the precision of your program's results if you are using **-O4** or higher, you should specify **-qhot=novector** if the change in precision is unacceptable.

## **Defaults**

The default is **-qnohot**.

Specifying **-qhot** without suboptions implies **-qhot=nosimd**, **-qhot=noarraypad**, **-qhot=level=1** and **-qhot=vector**. The **-qhot** option is also implied by **-qsmp**, **-O4**, and **-O5**. The **-qhot=level=0** option is implied by **-O3**.

When you specify **-qhot=level=0**, the default is set to **novector**, **nosimd** and **noarraypad**.

## **Examples**

The following example turns on the **-qhot=vector** option.

```
xlf95 -qhot=vector t.f
```

## **Related information**

- “-qarch option” on page 114
- “-C option” on page 89
- “-qstrict option” on page 230
- “-qsmp option” on page 222
- “-qenablevmx option” on page 141
- “-O option” on page 100
- *Directives for loop optimization in the XL Fortran Language Reference*
- *Benefits of High Order Transformation in the XL Fortran Optimization and Programming Guide*

## -qieee option

### Syntax

`-qieee={Near | Minus | Plus | Zero}`  
`IEEE({Near | Minus | Plus | Zero})`

Specifies the rounding mode for the compiler to use when it evaluates constant floating-point expressions at compile time.

### Arguments

The choices are:

|              |                                        |
|--------------|----------------------------------------|
| <b>Near</b>  | Round to nearest representable number. |
| <b>Minus</b> | Round toward minus infinity.           |
| <b>Plus</b>  | Round toward plus infinity.            |
| <b>Zero</b>  | Round toward zero.                     |

This option is intended for use in combination with the XL Fortran subroutine `fpsets` or some other method of changing the rounding mode at run time. It sets the rounding mode that is used for compile-time arithmetic (for example, evaluating constant expressions such as `2.0/3.5`). By specifying the same rounding mode for compile-time and run-time operations, you can avoid inconsistencies in floating-point results.

**Note:** Compile-time arithmetic is most extensive when you also specify the `-O` option.

If you change the rounding mode to other than the default (round-to-nearest) at run time, be sure to also specify `-qfloat=rrm` to turn off optimizations that only apply in the default rounding mode.

### Related information

- *Selecting the rounding mode* in the *XL Fortran Optimization and Programming Guide*
- “`-O` option” on page 100
- “`-qfloat` option” on page 152

## **-qinit option**

### **Syntax**

```
-qinit=f90ptr
INIT(F90PTR)
```

Makes the initial association status of pointers disassociated. Note that this applies to Fortran 90 and above.

You can use this option to help locate and fix problems that are due to using a pointer before you define it.

### **Related information**

See *Pointer Association* in the *XL Fortran Language Reference*.

## -qinitauto option

### Syntax

-qinitauto[=*hex\_value*] | **-qnoinitauto**

Initializes each byte or word (4 bytes) of storage for automatic variables to a specific value, depending on the length of the *hex\_value*. This helps you to locate variables that are referenced before being defined. For example, by using both the **-qinitauto** option to initialize **REAL** variables with a NaNS value and the **-qfltrap** option, it is possible to identify references to uninitialized **REAL** variables at run time. Prior to XL Fortran Version 5.1.1, you could only use this option to initialize each byte of storage.

Setting *hex\_value* to zero ensures that all automatic variables are cleared before being used. Some programs assume that variables are initialized to zero and do not work when they are not. Other programs may work if they are not optimized but fail when they are optimized. Typically, setting all the variables to all zero bytes prevents such run-time errors. It is better to locate the variables that require zeroing and insert code in your program to do so than to rely on this option to do it for you. Using this option will generally zero more things than necessary and may result in slower programs.

To locate and fix these errors, set the bytes to a value other than zero, which will consistently reproduce incorrect results. This method is especially valuable in cases where adding debugging statements or loading the program into a symbolic debugger makes the error go away.

Setting *hex\_value* to **FF** (255) gives **REAL** and **COMPLEX** variables an initial value of “negative not a number”, or -NaNQ. Any operations on these variables will also result in NaNQ values, making it clear that an uninitialized variable has been used in a calculation.

This option can help you to debug programs with uninitialized variables in subprograms; for example, you can use it to initialize **REAL** variables with a NaNS value. You can initialize 8-byte **REAL** variables to double-precision NaNS values by specifying an 8-digit hexadecimal number, that, when repeated, has a double-precision NaNS value. For example, you could specify a number such as 7FBFFFFF, that, when stored in a **REAL(4)** variable, has a single-precision NaNS value. The value 7FF7FFFF, when stored in a **REAL(4)** variable, has a single-precision NaNQ value. If the same number is stored twice in a **REAL(8)** variable (7FF7FFFF7FF7FFFF), it has a double-precision NaNS value.

### Arguments

- The *hex\_value* is a 1-digit to 8-digit hexadecimal (0-F) number.
- To initialize each byte of storage to a specific value, specify 1 or 2 digits for the *hex\_value*. If you specify only 1 digit, the compiler pads the *hex\_value* on the left with a zero.
- To initialize each word of storage to a specific value, specify 3 to 8 digits for the *hex\_value*. If you specify more than 2 but fewer than 8 digits, the compiler pads the *hex\_value* on the left with zeros.
- In the case of word initialization, if automatic variables are not a multiple of 4 bytes in length, the *hex\_value* may be truncated on the left to fit. For example, if you specify 5 digits for the *hex\_value* and an automatic variable is only 1 byte long, the compiler truncates the 3 digits on the left-hand side of the *hex\_value* and assigns the two right-hand digits to the variable.

- You can specify alphabetic digits as either upper- or lower-case.

### Defaults

- By default, the compiler does not initialize automatic storage to any particular value. However, it is possible that a region of storage contains all zeros.
- If you do not specify a *hex\_value* suboption for **-qinitauto**, the compiler initializes the value of each byte of automatic storage to zero.

### Restrictions

- Equivalenced variables, structure components, and array elements are not initialized individually. Instead, the entire storage sequence is initialized collectively.

### Examples

The following example shows how to perform word initialization of automatic variables:

```
subroutine sub()
integer(4), automatic :: i4
character, automatic :: c
real(4), automatic :: r4
real(8), automatic :: r8
end subroutine
```

When you compile the code with the following option, the compiler performs word initialization, as the *hex\_value* is longer than 2 digits:

```
-qinitauto=0cf
```

The compiler initializes the variables as follows, padding the *hex\_value* with zeros in the cases of the *i4*, *r4*, and *r8* variables and truncating the first hexadecimal digit in the case of the *c* variable:

| Variable | Value            |
|----------|------------------|
| i4       | 000000CF         |
| c        | CF               |
| r4       | 000000CF         |
| r8       | 000000CF000000CF |

### Related information

See “-qflttrap option” on page 154 and the section on the **AUTOMATIC** directive in the *XL Fortran Language Reference*.

## -qintlog option

### Syntax

```
-qintlog | -qnointlog
INTLOG | NOINTLOG
```

Specifies that you can mix integer and logical data entities in expressions and statements. Logical operators that you specify with integer operands act upon those integers in a bit-wise manner, and integer operators treat the contents of logical operands as integers.

### Restrictions

The following operations do not allow the use of logical variables:

- **ASSIGN** statement variables
- Assigned **GOTO** variables
- **DO** loop index variables
- Implied-**DO** loop index variables in **DATA** statements
- Implied-**DO** loop index variables in either input and output or array constructors
- Index variables in **FORALL** constructs

### Related information

- -qport=clogicals option.

### Examples

```
INTEGER I, MASK, LOW_ORDER_BYTE, TWOS_COMPLEMENT
I = 32767
MASK = 255
! Find the low-order byte of an integer.
LOW_ORDER_BYTE = I .AND. MASK
! Find the twos complement of an integer.
TWOS_COMPLEMENT = .NOT. I
END
```

### Related information

You can also use the intrinsic functions **IAND**, **IOR**, **IEOR**, and **NOT** to perform bitwise logical operations.

## -qintsize option

### Syntax

`-qintsize=bytes`  
`INTSIZE(bytes)`

Sets the size of default **INTEGER** and **LOGICAL** data entities (that is, those for which no length or kind is specified).

### Background information

The specified size<sup>1</sup> applies to these data entities:

- **INTEGER** and **LOGICAL** specification statements with no length or kind specified.
- **FUNCTION** statements with no length or kind specified.
- Intrinsic functions that accept or return default **INTEGER** or **LOGICAL** arguments or return values unless you specify a length or kind in an **INTRINSIC** statement. Any specified length or kind must agree with the default size of the return value.
- Variables that are implicit integers or logicals.
- Integer and logical literal constants with no kind specified. If the value is too large to be represented by the number of bytes that you specified, the compiler chooses a size that is large enough. The range for 2-byte integers is  $-(2^{*15})$  to  $2^{*15}-1$ , for 4-byte integers is  $-(2^{*31})$  to  $2^{*31}-1$ , and for 8-byte integers is  $-(2^{*63})$  to  $2^{*63}-1$ .
- Typeless constants in integer or logical contexts.

Allowed sizes for *bytes* are:

- 2
- 4 (the default)
- 8

This option is intended to allow you to port programs unchanged from systems that have different default sizes for data. For example, you might need `-qintsize=2` for programs that are written for a 16-bit microprocessor or `-qintsize=8` for programs that are written for a CRAY computer. The default value of 4 for this option is suitable for code that is written specifically for many 32-bit computers. Note that specifying the `-q64` compiler option does not affect the default setting for `-qintsize`.

### Restrictions

This option is not intended as a general-purpose method for increasing the sizes of data entities. Its use is limited to maintaining compatibility with code that is written for other systems.

You might need to add **PARAMETER** statements to give explicit lengths to constants that you pass as arguments.

---

1. In Fortran 90/95 terminology, these values are referred to as *kind type parameters*.

## Examples

In the following example, note how variables, literal constants, intrinsics, arithmetic operators, and input/output operations all handle the changed default integer size.

```
@PROCESS INTSIZE(8)
 PROGRAM INTSIZETEST
 INTEGER I
 I = -9223372036854775807 ! I is big enough to hold this constant.
 J = ABS(I) ! So is implicit integer J.
 IF (I .NE. J) THEN
 PRINT *, I, '.NE.', J
 END IF
 END
```

The following example only works with the default size for integers:

```
CALL SUB(17)
END

SUBROUTINE SUB(I)
 INTEGER(4) I ! But INTSIZE may change "17"
 ! to INTEGER(2) or INTEGER(8).

 ...
END
```

If you change the default value, you must either declare the variable **I** as **INTEGER** instead of **INTEGER(4)** or give a length to the actual argument, as follows:

```
@PROCESS INTSIZE(8)
 INTEGER(4) X
 PARAMETER(X=17)
 CALL SUB(X) ! Use a parameter with the right length, or
 CALL SUB(17_4) ! use a constant with the right kind.
END
```

## Related information

See “-qrealsize option” on page 210 and *Type Parameters and Specifiers* in the *XL Fortran Language Reference*.

## -qipa option

### Syntax

`-qipa[=suboptions] | -qnoipa`

Enhances **-O** optimization by doing detailed analysis across procedures (interprocedural analysis or IPA).

You must also specify the **-O**, **-O2**, **-O3**, **-O4**, or **-O5** option when you specify **-qipa**. (Specifying the **-O5** option is equivalent to specifying the **-O4** option plus **-qipa=level=2**.) For additional performance benefits, you can also specify the **-Q** option. The **-qipa** option extends the area that is examined during optimization and inlining from a single procedure to multiple procedures (possibly in different source files) and the linkage between them.

You can fine-tune the optimizations that are performed by specifying suboptions.

To use this option, the necessary steps are:

1. Do preliminary performance analysis and tuning before compiling with the **-qipa** option. This is necessary because interprocedural analysis uses a two-phase mechanism, a compile-time and a link-time phase, which increases link time. (You can use the **noobject** suboption to reduce this overhead.)
2. Specify the **-qipa** option on both the compile and link steps of the entire application or on as much of it as possible. Specify suboptions to indicate what assumptions to make about the parts of the program that are not compiled with **-qipa**. If your application contains C or C++ code compiled with IBM XL C/C++ compilers, you should also compile the C or C++ code with the **-qipa** option to allow for additional optimization opportunities at link time.)

During compilation, the compiler stores interprocedural analysis information in the **.o** file. During linking, the **-qipa** option causes a complete reoptimization of the entire application.

Note that if you specify this option with **-#**, the compiler does not display linker information subsequent to the IPA link step. This is because the compiler does not actually call IPA.

### Arguments

#### Compile-time suboptions

IPA uses the following suboptions during its compile-time phase:

#### object | **noobject**

Specifies whether to include standard object code in the object files. Specifying the **noobject** suboption can substantially reduce overall compilation time, by not generating object code during the first IPA phase. Note that if you specify **-S** with **noobject**, **noobject** will be ignored.

If compiling and linking are performed in the same step and you do not specify the **-S** or any listing option, **-qipa=noobject** is implied.

If your program contains object files created with the **noobject** suboption, you must use the **-qipa** option to compile any files containing an entry point (the main program for an executable program or an exported procedure for a library) before linking your program with **-qipa**.

#### Link-time suboptions

IPA uses the following suboptions during its link-time phase:

**clonearch=arch{,arch} | noclonearch**

Specifies the architectures for which multiple versions of the same instruction set are produced. Use this suboption if you require optimal performance on multiple differing machines running the same copy of your application.

The compiler generates a generic version of the instruction set based on the **-qarch** setting in effect, and if appropriate, clones specialized versions of the instruction set for the architectures you specify in the **clonearch** suboption. The compiler inserts code into your application to check for the processor architecture at run time. When run, the application's version of the instruction set that is best optimized for the run-time environment is selected.

*arch* is a comma-separated list of architecture values. The supported clonearch values are **pwr4**, **pwr5**, and **ppc970**. If you specify no value, an invalid value, or a value equal to the **-qarch** setting, no function versioning will be performed for this option.

**Notes:**

1. To ensure compatibility across multiple platforms, the **-qarch** value must be the subset of the architecture specified by **-qarch=clonearch=arch**.
2. When **-qcompact** is in effect, **-qarch=clonearch=arch** is disabled.

Table 18. Compatible architecture and clonearch settings

| <b>-qarch setting</b>                            | <b>Allowed clonearch value</b> |
|--------------------------------------------------|--------------------------------|
| com, ppc, pwr3, ppc64, ppcgr, ppc64gr, ppc64grsq | pwr4, pwr5, ppc970             |
| pwr4                                             | pwr5, ppc970                   |
| ppc64v                                           | ppc970                         |

**cloneproc=name{name} | nocloneproc[=name{name}]**

Specifies the name of the functions to clone for the architectures specified by the clonearch suboption. Where *name* is a comma-separated list of function names.

**Note:** If you do not specify **-qipa=clonearch=arch** or specify **-qipa=noclonearch**, **-qipa=cloneproc=name** and **-qipa=nocloneproc=name** have no effect.

**exits=procedure\_names**

Specifies a list of procedures, each of which always ends the program. The compiler can optimize calls to these procedures (for example, by eliminating save/restore sequences), because the calls never return to the program. These procedures must not call any other parts of the program that are compiled with **-qipa**.

**inline=inline-options**

The **-qipa=inline=** command can take a colon-separated list of inline options, as listed below:

**auto | noauto**

Specifies whether to automatically inline procedures.

**limit=number**

Changes the size limits that the **inline=auto** option uses to determine how much inline expansion to do. This established “limit” is the size below which the calling procedure must remain. *number* is the optimizer’s approximation of the number of bytes of code that will be generated. Larger values for this number allow the compiler to inline larger subprograms, more subprogram calls, or both. This argument is implemented only when **inline=auto** is on.

*procedure\_names*

Specifies a list of procedures to try to inline.

**threshold=number**

Specifies the upper size limit on procedures to be inlined, where *number* is a value as defined under the inline suboption “limit”. This argument is implemented only when “inline=auto” is on.

**Note:** By default, the compiler will try to inline all procedures, not just those that you specified with the **inline=procedure\_names** suboption. If you want to turn on inlining for only certain procedures, specify **inline=noauto** after you specify **inline=procedure\_names**. (You must specify the suboptions in this order.) For example, to turn off inlining for all procedures other than for **sub1**, specify **-qipa=inline=sub1:inline=noauto**.

**isolated=procedure\_names**

Specifies a comma-separated list of procedures that are not compiled with **-qipa**. Procedures that you specify as “isolated” or procedures within their call chains cannot refer directly to any global variable.

**level=level**

Determines the amount of interprocedural analysis and optimization that is performed:

- 0 Does only minimal interprocedural analysis and optimization.
- 1 Turns on inlining, limited alias analysis, and limited call-site tailoring.
- 2 Full interprocedural data flow and alias analysis. Specifying **-O5** is equivalent to specifying **-O4** and **-qipa=level=2**.

The default level is 1.

**list[=filename | short | long]**

Specifies an output listing file name during the link phase, in the event that an object listing has been requested using either the **-qlist** or the **-qipa=list** compiler option and allows the user to direct the type of output. If you do not specify the *filename* suboption, the default file name is “a.lst”.

If you specify **short**, the Object File Map, Source File Map, and Global Symbols Map sections are included. If you specify **long**, the preceding sections appear in addition to the Object Resolution Warnings, Object Reference Map, Inliner Report, and Partition Map sections.

If you specify the **-qipa** and **-qlist** options together, IPA generates an a.lst file that overwrites any existing a.lst file. If you have a source file named a.f, the IPA listing will overwrite the regular compiler listing a.lst. You can use the **list=filename** suboption to specify an alternative listing file name.

**lowfreq**=*procedure\_names*

Specifies a list of procedures that are likely to be called infrequently during the course of a typical program run. For example, procedures for initialization and cleanup might only be called once, and debugging procedures might not be called at all in a production-level program. The compiler can make other parts of the program faster by doing less optimization for calls to these procedures.

**malloc16** | **nomalloc16**

Informs the compiler that the dynamic memory allocation routines will return 16-byte aligned memory addresses. The compiler can then optimize the code based on that assertion.

In 64-bit mode, AIX always returns 16-byte aligned addresses and therefore by default, **-qipa=malloc16** is in effect. You can use **-qipa=nomalloc16** to override the default setting.

**Note:** You must make sure that the executables generated with **-qipa=malloc16** run in an environment in which dynamic memory allocations return 16-byte aligned addresses, otherwise, wrong results can be generated. For example, in 32-bit mode, addresses are not 16-byte aligned. In this case, you must set the **MALLOCALIGN=16** runtime option.

**missing**={**unknown** | **safe** | **isolated** | **pure**}

Specifies the interprocedural behavior of procedures that are not compiled with **-qipa** and are not explicitly named in an **unknown**, **safe**, **isolated**, or **pure** suboption. The default is to assume **unknown**, which greatly restricts the amount of interprocedural optimization for calls to those procedures.

**noinline**=*procedure\_names*

Specifies a list of procedures that are not to be inlined.

**partition**={**small** | **medium** | **large**}

Specifies the size of the regions within the program to analyze. Larger partitions contain more procedures, which result in better interprocedural analysis but require more storage to optimize. Reduce the partition size if compilation takes too long because of paging.

**pdfname**=[*filename*]

Specifies the name of the profile data file containing the **PDF** profiling information. If you do not specify a *filename*, the default file name is **\_\_pdf**. The profile is placed in the current working directory or in the directory that the **PDFDIR** environment variable names. This allows you to do simultaneous runs of multiple executables using the same **PDFDIR**. This is especially useful when tuning with **PDF** on dynamic libraries. (See “-qpdf option” on page 198 for more information on tuning optimizations.)

**pure**=*procedure\_names*

Specifies a list of procedures that are not compiled with **-qipa**. Any procedure that you specified as “pure” must be “isolated” and “safe”. It must not alter the internal state nor have side-effects, which are defined as potentially altering any data object visible to the caller.

**safe**=*procedure\_names*

Specifies a list of procedures that are not compiled with **-qipa**. Any procedure that you specified as “safe” may modify global variables and dummy arguments. No calls to procedures that are compiled with **-qipa** may be made from within a “safe” procedure’s call chain.

**stdexits** | **nostdexits**

Specifies that certain predefined routines can be optimized as with the **exits** suboption. The procedures are: **abort**, **exit**, **\_exit**, and **\_assert**.

**threads[=N]** | **nothreads**

**threads[=N]** runs the number of parallel threads that are available, or as specified by *N*. *N* must be a positive integer. **nothreads** does not run any parallel threads. This is equivalent to running one serial thread.

Specifying **-qipa=threads** can reduce IPA optimization time. The threads suboption allows the IPA optimizer to run portions of the optimization process in parallel threads, which can speed up the compilation process on multi-processor systems.

**unknown=procedure\_names**

Specifies a list of procedures that are not compiled with **-qipa**. Any procedure specified as “unknown” may make calls to other parts of the program compiled with **-qipa** and modify global variables and dummy arguments.

The primary use of **isolated**, **missing**, **pure**, **safe**, and **unknown** is to specify how much optimization can safely be performed on calls to library routines that are not compiled with **-qipa**.

The following compiler options have an effect on the link-time phase of **-qipa**:

**-qlibansi** | **-qnolibansi**

Assumes that all functions with the name of an ANSI C defined library function are, in fact, the library functions.

**-qlibessl** | **-qnolibessl**

Assumes that all functions with the name of an ESSL defined library function are, in fact, the library functions.

**-qlibposix** | **-qnolibposix**

Assumes that all functions with the name of a POSIX 1003.1 defined library function are, in fact, the system functions.

**-qthreaded**

Assumes that the compiler will attempt to generate thread-safe code.

**Applicable product levels**

This option is similar but not identical to the **-qipa** option of XL Fortran Version 3. If you have makefiles that already contain the **-qipa** option, modify them as needed to use the new suboption names.

**Rules**

Regular expressions are supported for the following suboptions:

- exits
- inline
- lowfreq
- noinline
- pure
- safe
- unknown

Syntax rules for regular expressions are described below.

Table 19. Regular expression syntax

| Expression        | Description                                                                                                                                                                                                                                                                            |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| string            | Matches any of the characters specified in string. For example, test will match testimony, latest, intestine.                                                                                                                                                                          |
| ^string           | Matches the pattern specified by string only if it occurs at the beginning of a line.                                                                                                                                                                                                  |
| string\$          | Matches the pattern specified by string only if it occurs at the end of a line.                                                                                                                                                                                                        |
| str.ing           | Matches any character. For example, t.st will match test, tast, tZst, and t1st.                                                                                                                                                                                                        |
| string\.\$        | The backslash (\) can be used to escape special characters so that you can match for the character. For example, if you want to find those lines ending with a period, the expression .\$ would show all lines that had at least one character. Specify \.\$ to escape the period (.). |
| [string]          | Matches any of the characters specified in string. For example, t[a-g123]st matches tast and test, but not t-st or tAst.                                                                                                                                                               |
| [^string]         | Does not match any of the characters specified in string. For example, t[^a-zA-Z]st matches t1st, t-st, and t,st but not test or tYst.                                                                                                                                                 |
| string*           | Matches zero or more occurrences of the pattern specified by string. For example, te*st will match tst, test, and teeeeeest.                                                                                                                                                           |
| string+           | Matches one or more occurrences of the pattern specified by string. For example, t(es)+t matches test, tesest, but not tt.                                                                                                                                                             |
| string?           | Matches zero or more occurrences of the pattern specified by string. For example, te?st matches either tst or test.                                                                                                                                                                    |
| string{m,n}       | Matches between m and n occurrence(s) of the pattern specified by string. For example, a{2} matches aa, b{1,4} matches b, bb, bbb, and bbbb.                                                                                                                                           |
| string1   string2 | Matches the pattern specified by either string1 or string2. For example, s   o matches both characters s and o.                                                                                                                                                                        |

Since only function names are being considered, the regular expressions are automatically bracketed with the ^ and \$ characters. For example, **-qipa=noninline=^foo\$** is equivalent to **-qipa=noninline=foo**. Therefore, **-qipa=noninline=bar** ensures that **bar** is never inlined but **bar1**, **teebar**, and **barrel** may be inlined.

### Examples

The following example shows how you might compile a set of files with interprocedural analysis:

```
xlf95 -O -qipa f.f
xlf95 -c -O3 *.f -qipa=noobject
xlf95 -o product *.o -qipa -O
```

The following example shows how you might link these same files with interprocedural analysis, using regular expressions to improve performance. This

example assumes that function **user\_abort** exits the program, and that routines **user\_trace1**, **user\_trace2**, and **user\_trace3** are rarely called.

```
xlf95 -o product *.o -qipa=exit=user_abort:lowfreq=user_trace[123] -0
```

### **Related information**

See the “-O option” on page 100, “-p option” on page 103, and “-Q option” on page 104.

## **-qkeepparm option**

### **Syntax**

`-qkeepparm` | `-qnokeepparm`

### **Background information**

A procedure usually stores its incoming parameters on the stack at the entry point. When you compile code with optimization, however, the optimizer may remove the stores into the stack if it sees opportunities to do so.

Specifying the **-qkeepparm** compiler option ensures that the parameters are stored on the stack even when optimizing. This may negatively impact execution performance. This option then provides access to the values of incoming parameters to tools, such as debuggers, simply by preserving those values on the stack.

## -qlanglvl option

### Syntax

```
-qlanglvl={suboption}
LANGVLV({suboption})
```

Determines which language standard (or superset, or subset of a standard) to consult for nonconformance. It identifies nonconforming source code and also options that allow such nonconformances.

### Rules

The compiler issues a message with severity code **L** if it detects syntax that is not allowed by the language level that you specified.

### Arguments

|                        |                                                                                                                                                                        |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>77std</b>           | Accepts the language that the ANSI FORTRAN 77 standard specifies and reports anything else as an error.                                                                |
| <b>90std</b>           | Accepts the language that the ISO Fortran 90 standard specifies and reports anything else as an error.                                                                 |
| <b>90pure</b>          | The same as <b>90std</b> except that it also reports errors for any obsolescent Fortran 90 features used.                                                              |
| <b>90ext</b>           | Obsolete suboption that is equivalent to <b>extended</b> .                                                                                                             |
| <b>95std</b>           | Accepts the language that the ISO Fortran 95 standard specifies and reports anything else as an error.                                                                 |
| <b>95pure</b>          | The same as <b>95std</b> except that it also reports errors for any obsolescent Fortran 95 features used.                                                              |
| <b>2003std</b>         | Accepts the language that the ISO Fortran 95 standard specifies, as well as all Fortran 2003 features supported by XL Fortran, and reports anything else as an error.  |
| <b>2003pure</b>        | The same as <b>2003std</b> except that it also reports errors for any obsolescent Fortran 2003 features used.                                                          |
| <b><u>extended</u></b> | Accepts the full Fortran 95 language standard, all Fortran 2003 features supported by XL Fortran, and all extensions, effectively turning off language-level checking. |

### Defaults

The default is **-qlanglvl=extended**. Prior to XL Fortran Version 6.1, the default was **-qlanglvl=90ext**. The **90ext** suboption accepts the full Fortran 90 language standard plus all extensions (now including the Fortran 95 standard) and is equivalent to **extended**. However, the **90ext** suboption is now obsolete, and to avoid problems in the future, you should start using the **extended** suboption as soon as possible.

## Restrictions

The **-qflag** option can override this option.

## Examples

The following example contains source code that conforms to a mixture of Fortran standards:

```
!-----
! in free source form
program tt
 integer :: a(100,100), b(100), i
 real :: x, y
 ...
 goto (10, 20, 30), i
10 continue
 pause 'waiting for input'

20 continue
 y= gamma(x)

30 continue
 b = maxloc(a, dim=1, mask=a .lt 0)

end program
!-----
```

The following chart shows examples of how some **-qlanglvl** suboptions affect this sample program:

| <b>-qlanglvl Suboption Specified</b> | <b>Result</b>                                                                                        | <b>Reason</b>                                                                                 |
|--------------------------------------|------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| <b>95pure</b>                        | Flags <b>PAUSE</b> statement<br>Flags computed <b>GOTO</b> statement<br>Flags <b>GAMMA</b> intrinsic | Deleted feature in Fortran 95<br>Obsolescent feature in Fortran 95<br>Extension to Fortran 95 |
| <b>95std</b>                         | Flags <b>PAUSE</b> statement<br>Flags <b>GAMMA</b> intrinsic                                         | Deleted feature in Fortran 95<br>Extension to Fortran 95                                      |
| <b>extended</b>                      | No errors flagged                                                                                    |                                                                                               |

## Related information

See “-qflag option” on page 151, “-qhalt option” on page 158, and “-qsaa option” on page 215.

The **langlvl** run-time option, which is described in “Setting run-time options” on page 40, helps to locate run-time extensions that cannot be checked for at compile time.

## **-qlargepage option**

### **Syntax**

`-qlargepage` | `-qno1argepage`

Indicates to the compiler that a program, designed to execute in a large page memory environment, can take advantage of large 16 MB pages provided on POWER4 and higher based systems. When using `-qlargepage` with a program designed for a large page environment, an increase in performance can occur. See *AIX Performance Management Guide* for more information on using large page support.

**Note:** When using AIX 5.1, performance degradation can occur if there are too many programs attempting to access large pages at the same time. Performance degradation can also occur if you attempt to use `-qlargepage` without meeting the hardware requirements. Use this option with discretion.

The `-qlargepage` compiler option only takes effect with an optimization level that turns on the optimizer; a higher optimization level may do more.

## **-qlibansi option**

### **Syntax**

`-qlibansi` | `-qno1ibansi`

Assumes that all functions with the name of an ANSI C library function are, in fact, the system functions.

This option will allow the optimizer to generate better code because it will know about the behavior of a given function, such as whether or not it has any side effects.

**Note:** Do not use this option if your application contains your own version of a C library function that is incompatible with the standard one

### **Related information**

See “-qipa option” on page 169.

## **-qlibessl option**

### **Related information**

See “-qipa option” on page 169.

## **-qlibposix option**

### **Related information**

See “-qipa option” on page 169.

## **-qlinedebug option**

### **Syntax**

`-qlinedebug` | `-qno1inedebug`  
`LINEDeBUG` | `NOLINEDeBUG`

Generates line number and source file name information for the debugger.

This option produces minimal debugging information, so the resulting object size is smaller than that produced if the `-g` debugging option is specified. You can use the debugger to step through the source code, but you will not be able to see or query variable information. The traceback table, if generated, will include line numbers.

As with `-g`, the higher the optimization level used, the more likely that debug information, including line numbers, will be misleading.

If you specify the `-qlinedebug` option, the inlining option defaults to `-Q!` (no functions are inlined).

The `-qno1inedebug` option has no effect on `-g`.

### **Related information**

- “`-g` option” on page 94
- “`-O` option” on page 100

## **-qlist option**

### **Syntax**

`-qlist[=offset | nooffset] | -qno list`  
`LIST[([NO]OFFSET)] | NOLIST`

Produces a compiler listing that includes an object listing.

You can use the object listing to help understand the performance characteristics of the generated code and to diagnose execution problems.

If you specify the **-qipa** and **-qlist** options together, IPA generates an `a.lst` file that overwrites any existing `a.lst` file. If you have a source file named `a.f`, the IPA listing will overwrite the regular compiler listing `a.lst`. To avoid this, use the `list=filename` suboption of **-qipa** to generate an alternative listing.

### **Notes**

The **-qlist=offset** option is relevant only if there are multiple procedures in a compilation unit. For example, this may occur if nested procedures are used in a program.

If you specify **-qlist=offset**, the listing will show the offset from the start of the procedure rather than the offset from the start of code generation.

The offset of the PDEF header will contain the offset from the start of the text area. This allows any program reading the `.lst` file to add the value of the PDEF and the line in question, and produce the same value whether **-qlist=offset** or **-qlist=nooffset** is specified.

Specifying the **-qlist** option implies **-qlist=nooffset**.

### **Related information**

- “Options that control listings and messages” on page 62
- “Object section” on page 299
- “-S option” on page 260
- Program units and procedures in the *XL Fortran Language Reference*

## -qlistopt option

### Syntax

-qlistopt | **-qnolistopt**  
LISTOPT | **NOLISTOPT**

Determines whether to show the setting of every compiler option in the listing file or only selected options. These selected options include those specified on the command line or directives plus some that are always put in the listing.

You can use the option listing during debugging to check whether a problem occurs under a particular combination of compiler options or during performance testing to record the optimization options in effect for a particular compilation.

### Rules

Options that are always displayed in the listing are:

- All “on/off” options that are on by default: for example, **-qobject**
- All “on/off” options that are explicitly turned off through the configuration file, command-line options, or **@PROCESS** directives
- All options that take arbitrary numeric arguments (typically sizes)
- All options that have multiple suboptions

### Related information

See “Options that control listings and messages” on page 62 and “Options section” on page 295.

## **-qlog4 option**

### **Syntax**

`-qlog4` | `-qnolog4`  
`LOG4` | `NOLOG4`

Specifies whether the result of a logical operation with logical operands is a **LOGICAL(4)** or is a **LOGICAL** with the maximum length of the operands.

You can use this option to port code that was originally written for the IBM VS FORTRAN compiler.

### **Arguments**

`-qlog4` makes the result always a **LOGICAL(4)**, while `-qnolog4` makes it depend on the lengths of the operands.

### **Restrictions**

If you use `-qintsize` to change the default size of logicals, `-qlog4` is ignored.

## **-qmaxmem option**

### **Syntax**

`-qmaxmem=Kbytes`  
`MAXMEM(Kbytes)`

Limits the amount of memory that the compiler allocates while performing specific, memory-intensive optimizations to the specified number of kilobytes. A value of -1 allows optimization to take as much memory as it needs without checking for limits.

### **Defaults**

At the **-O2** optimization level, the default **-qmaxmem** setting is 8192 KB. At the **-O3** optimization level, the default setting is unlimited (-1).

### **Rules**

If the specified amount of memory is insufficient for the compiler to compute a particular optimization, the compiler issues a message and reduces the degree of optimization.

This option has no effect except in combination with the **-O** option.

When compiling with **-O2**, you only need to increase the limit if a compile-time message instructs you to do so. When compiling with **-O3**, you might need to establish a limit if compilation stops because the machine runs out of storage; start with a value of 8192 or higher, and decrease it if the compilation continues to require too much storage.

### **Notes:**

1. Reduced optimization does not necessarily mean that the resulting program will be slower. It only means that the compiler cannot finish looking for opportunities to improve performance.
2. Increasing the limit does not necessarily mean that the resulting program will be faster. It only means that the compiler is better able to find opportunities to improve performance if they exist.
3. Setting a large limit has no negative effect when compiling source files for which the compiler does not need to use so much memory during optimization.
4. As an alternative to raising the memory limit, you can sometimes move the most complicated calculations into procedures that are then small enough to be fully analyzed.
5. Not all memory-intensive compilation stages can be limited.
6. Only the optimizations done for **-O2** and **-O3** can be limited; **-O4** and **-O5** optimizations cannot be limited.
7. The **-O4** and **-O5** optimizations may also use a file in the `/tmp` directory. This is not limited by the **-qmaxmem** setting.
8. Some optimizations back off automatically if they would exceed the maximum available address space, but not if they would exceed the paging space available at that time, which depends on machine workload.

## Restrictions

Depending on the source file being compiled, the size of subprograms in the source code, the machine configuration, and the workload on the system, setting the limit too high might fill up the paging space. In particular, a value of -1 can fill up the storage of even a well-equipped machine.

## Related information

- “-O option” on page 100
- *Optimizing XL compiler programs* in the *XL Fortran Optimization and Programming Guide*

## -qmbcs option

### Syntax

-qmbcs | -qnombcs  
MBCS | NOMBCS

Indicates to the compiler whether character literal constants, Hollerith constants, H edit descriptors, and character string edit descriptors can contain Multibyte Character Set (MBCS) or Unicode characters.

This option is intended for applications that must deal with data in a multibyte language, such as Japanese.

To process the multibyte data correctly at run time, set the locale (through the **LANG** environment variable or a call to the **libc setlocale** routine) to the same value as during compilation.

### Rules

Each byte of a multibyte character is counted as a column.

### Restrictions

To read or write Unicode data, set the locale value to **UNIVERSAL** at run time. If you do not set the locale, you might not be able to interchange data with Unicode-enabled applications.

## **-qmixed option**

### **Syntax**

`-qmixed` | `-qnomixed`  
`MIXED` | `NOMIXED`

This is the long form of the “-U option” on page 262.

## **-qmoddir option**

### **Syntax**

`-qmoddir=directory`

Specifies the location for any module (**.mod**) files that the compiler writes.

### **Defaults**

If you do not specify **-qmoddir**, the **.mod** files are placed in the current directory.

To read the **.mod** files from this directory when compiling files that reference the modules, use the “-I option” on page 95.

### **Related information**

See “XL Fortran Output files” on page 25.

Modules are a Fortran 90/95 feature and are explained in the Modules section of the *XL Fortran Language Reference*.

## **-qmodule option**

### **Syntax**

`-qmodule=mangle81`

Specifies that the compiler should use the XL Fortran Version 8.1 naming convention for non-intrinsic module files.

This option allows you to produce modules and their associated object files with the Version 10.1 compiler and link these object files with others compiled with the Version 8.1 compiler, or earlier.

Use this option only if you need to link applications that were compiled with the Version 8.1 compiler, or earlier. It is recommended that you avoid using this compiler switch and rebuild old code and modules with the new version of the compiler, if possible. Doing so will avoid any naming conflicts between your modules and intrinsic compiler modules.

### **Related information**

- Modules section in the *XL Fortran Language Reference*. (Modules are a Fortran 90/95 feature.)
- *Conventions for XL Fortran external names* in the *XL Fortran Optimization and Programming Guide*
- “Avoiding naming conflicts during linking” on page 36

## **-qnoprint option**

### **Syntax**

`-qnoprint`

Prevents the compiler from creating the listing file, regardless of the settings of other listing options.

Specifying **-qnoprint** on the command line enables you to put other listing options in a configuration file or on **@PROCESS** directives and still prevent the listing file from being created.

### **Rules**

A listing file is usually created when you specify any of the following options: **-qattr**, **-qlist**, **-qlistopt**, **-qphsinfo**, **-qsource**, **-qreport**, or **-qxref**. **-qnoprint** prevents the listing file from being created by changing its name to **/dev/null**, a device that discards any data that is written to it.

### **Related information**

See “Options that control listings and messages” on page 62.

## -qnullterm option

### Syntax

```
-qnullterm | -qnonullterm
NULLTERM | NONULLTERM
```

Appends a null character to each character constant expression that is passed as a dummy argument, to make it more convenient to pass strings to C functions.

This option allows you to pass strings to C functions without having to add a null character to each string argument.

### Background information

This option affects arguments that are composed of any of the following objects: basic character constants; concatenations of multiple character constants; named constants of type character; Hollerith constants; binary, octal, or hexadecimal typeless constants when an interface block is available; or any character expression composed entirely of these objects. The result values from the **CHAR** and **ACHAR** intrinsic functions also have a null character added to them if the arguments to the intrinsic function are initialization expressions.

### Rules

This option does not change the length of the dummy argument, which is defined by the additional length argument that is passed as part of the XL Fortran calling convention.

### Restrictions

This option affects those arguments that are passed with or without the **%REF** built-in function, but it does not affect those that are passed by value. This option does not affect character expressions in input and output statements.

### Examples

Here are two calls to the same C function, one with and one without the option:

```
@PROCESS NONULLTERM
SUBROUTINE CALL_C_1
 CHARACTER*9, PARAMETER :: HOME = "/home/luc"
! Call the libc routine mkdir() to create some directories.
 CALL mkdir ("/home/luc/testfiles\0", %val(448))
! Call the libc routine unlink() to remove a file in the home directory.
 CALL unlink (HOME // "/.hushlogin" // CHAR(0))
END SUBROUTINE

@PROCESS NULLTERM
SUBROUTINE CALL_C_2
 CHARACTER*9, PARAMETER :: HOME = "/home/luc"
! With the option, there is no need to worry about the trailing null
! for each string argument.
 CALL mkdir ("/home/luc/testfiles", %val(448))
 CALL unlink (HOME // "/.hushlogin")
END SUBROUTINE
!
```

### Related information

See *Passing character types between languages* in the *XL Fortran Optimization and Programming Guide*.

## **-qobject option**

### **Syntax**

**-qOBJect** | **-qNOOBJect**  
**OBJect** | **NOOBJect**

Specifies whether to produce an object file or to stop immediately after checking the syntax of the source files.

When debugging a large program that takes a long time to compile, you might want to use the **-qnoobject** option. It allows you to quickly check the syntax of a program without incurring the overhead of code generation. The **.lst** file is still produced, so you can get diagnostic information to begin debugging.

After fixing any program errors, you can change back to the default (**-qobject**) to test whether the program works correctly. If it does not work correctly, compile with the **-g** option for interactive debugging.

### **Restrictions**

The **-qhalt** option can override the **-qobject** option, and **-qnoobject** can override **-qhalt**.

### **Related information**

See “Options that control listings and messages” on page 62 and “Object section” on page 299.

“The compiler phases” on page 303 gives some technical information about the compiler phases.

## **-qonetrip option**

### **Syntax**

-qonetrip | -qnoonetrip  
ONETRIP | NOONETRIP

This is the long form of the “-1 option” on page 78.

## **-qoptimize option**

### **Syntax**

`-qOPTimize[=level] | -qNOOPTimize`  
`OPTimize[(level)] | NOOPTimize`

This is the long form of the “-O option” on page 100.

## -qpdf option

### Syntax

-qpdf{1|2}

Tunes optimizations through *profile-directed feedback* (PDF), where results from sample program execution are used to improve optimization near conditional branches and in frequently executed code sections.

To use PDF, follow these steps:

1. Compile some or all of the source files in a program with the **-qpdf1** option. You need to specify the **-O2** option, or preferably the **-O3**, **-O4**, or **-O5** option, for optimization. Pay special attention to the compiler options that you use to compile the files, because you will need to use the same options later.

In a large application, concentrate on those areas of the code that can benefit most from optimization. You do not need to compile all of the application's code with the **-qpdf1** option.

If you want to avoid full recompilation in the **-qpdf2** step, first compile with **-qpdf1** but with no linking, so that the object files are saved. Then link the object files into an executable. To use this procedure, issue commands similar to the following:

```
xlf -c -qpdf1 -O2 file1.f file2.f
xlf -qpdf1 -O2 file1.o file2.o
```

2. Run the program all the way through using a typical data set. The program records profiling information when it finishes. You can run the program multiple times with different data sets, and the profiling information is accumulated to provide an accurate count of how often branches are taken and blocks of code are executed.

**Important:** Use data that is representative of the data that will be used during a normal run of your finished program.

3. Recompile and relink your program using the same compiler options as before, but change **-qpdf1** to **-qpdf2**. Remember that **-L**, **-I**, and some others are linker options, and you can change them at this point. In this second compilation, the accumulated profiling information is used to fine-tune the optimizations. The resulting program contains no profiling overhead and runs at full speed.

Alternatively, if you saved the object files from 1, you can simply relink them. For example, issue the following command:

```
xlf -qpdf2 -O2 file1.o file2.o
```

This can save considerable time and help fine tune large applications for optimization. You can create and test different flavors of PDF optimized binaries by using different options on the **-qpdf2** pass.

For best performance, use the **-O3**, **-O4**, or **-O5** option with all compilations when you use PDF (as in the example above). If your application contains C or C++ code compiled with IBM XL C/C+ compilers, you can achieve additional PDF optimization by specifying the **-qpdf1** and **-qpdf2** options available on those compilers. Combining **-qpdf1**/**-qpdf2** and **-qipa** or **-O5** options (that is, link with IPA) on all Fortran and C/C++ code will lead to maximum PDF information being available for optimization.

### Rules

The profile is placed in the current working directory or in the directory that the **PDFDIR** environment variable names, if that variable is set.

To avoid wasting compilation and execution time, make sure that the **PDFDIR** environment variable is set to an absolute path. Otherwise, you might run the application from the wrong directory, and it will not be able to locate the profile data files. When that happens, the program may not be optimized correctly or may be stopped by a segmentation fault. A segmentation fault might also happen if you change the value of the **PDFDIR** variable and execute the application before finishing the PDF process.

## Background information

Because this option requires compiling the entire application twice, it is intended to be used after other debugging and tuning is finished, as one of the last steps before putting the application into production.

## Restrictions

- PDF optimizations also require at least the **-O2** optimization level.
- You must compile the main program with PDF for profiling information to be collected at run time.
- Do not compile or run two different applications that use the same **PDFDIR** directory at the same time, unless you have used the **-qipa=pdfname** suboption to distinguish the sets of profiling information.
- You must use the same set of compiler options at all compilation steps for a particular program. Otherwise, PDF cannot optimize your program correctly and may even slow it down. All compiler settings must be the same, including any supplied by configuration files.
- The **-qpdf** option affects only programs whose source files have been compiled using XL Fortran Version 7.1.1 or later. Avoid mixing object files that were compiled with the **-qpdf** option on previous compilers, with those compiled with the **-qpdf** option on XL Fortran compilers at version 7.1.1 or later.
- If **-qipa** is not invoked either directly or through other options, **-qpdf1** and **-qpdf2** will invoke the **-qipa=level=0** option.
- If you do compile a program with **-qpdf1**, remember that it will generate profiling information when it runs, which involves some performance overhead. This overhead goes away when you recompile with **-qpdf2** or with no PDF at all.

The following commands, in the directory **/usr/lpp/xlf/bin**, are available for managing the **PDFDIR** directory:

**cleanpdf** [*pathname*]    Removes all profiling information from the *pathname* directory; or if *pathname* is not specified, from the **PDFDIR** directory; or if **PDFDIR** is not set, from the current directory.

Removing the profiling information reduces the run-time overhead if you change the program and then go through the PDF process again.

Run this program after compiling with **-qpdf2** or after finishing with the PDF process for a particular application. If you continue using PDF with an application after running **cleanpdf**, you must recompile all the files with **-qpdf1**.

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mergepdf                     | <p>Generates a single pdf record from 2 or more input pdf records. All pdf records must come from the same executable.</p> <p><b>mergepdf</b> automatically scales each pdf record (that is, file) based on its "weight". A scaling ratio can be specified for each pdf record, so that more important training runs can be weighted heavier than less important ones. The syntax for <b>mergepdf</b> is:</p> <pre>mergepdf [-r1] record1 [-r2] record2 ... -outputrecname [-n] [-v]</pre> <p>where:</p> <ul style="list-style-type: none"> <li><b>-r</b>        The scaling ratio for the pdf record. If <b>-r</b> is not specified for an input record, the default ratio is 1.0 and no external scaling is applied. The scaling ratio must be greater than or equal to zero, and can be a floating point number or an integer.</li> <li><b>record</b>    The input file, or the directory that contains the pdf profile.</li> <li><b>-o output_recordname</b><br/>              The pdf output directory name, or a file name that <b>mergepdf</b> will write the merged pdf record to. If a directory is specified, it must exist before you run the command.</li> <li><b>-n</b>        Do not normalize the pdf records. By default, records are normalized based on an internally calculated ratio for each profile before applying any user-specified ratio with <b>-r</b>. When <b>-n</b> is specified, the pdf records are scaled by the user-specified ratio <b>-r</b>. If <b>-r</b> is not specified, the pdf records are not scaled at all.</li> <li><b>-v</b>        Verbose mode displays the internal and user-specified scaling ratio used.</li> </ul> |
| resetpdf [ <i>pathname</i> ] | Same as <b>cleanpdf</b> [ <i>pathname</i> ], described above. This command is provided for compatibility with the previous version.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| showpdf                      | Displays the call and block counts for all procedures executed in a program run. To use this command, you must first compile your application specifying both <b>-qpdf1</b> and <b>-qshowpdf</b> compiler options                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

## Examples

Here is a simple example:

```
Set the PDFDIR variable.
export PDFDIR=$HOME/project_dir
Compile all files with -qpdf1.
xlf95 -qpdf1 -O3 file1.f file2.f file3.f
Run with one set of input data.
a.out <sample.data
Recompile all files with -qpdf2.
xlf95 -qpdf2 -O3 file1.f file2.f file3.f
The program should now run faster than without PDF if
the sample data is typical.
```

Here is a more elaborate example:

```
Set the PDFDIR variable.
export PDFDIR=$HOME/project_dir
Compile most of the files with -qpdf1.
xlf95 -qpdf1 -O3 -c file1.f file2.f file3.f
This file is not so important to optimize.
xlf95 -c file4.f
Non-PDF object files such as file4.o can be linked in.
xlf95 -qpdf1 -O3 file1.o file2.o file3.o file4.o
```

```

Run several times with different input data.
a.out <polar_orbit.data
a.out <elliptical_orbit.data
a.out <geosynchronous_orbit.data
Do not need to recompile the source of non-PDF object files (file4.f).
xlf95 -qpdf2 -O3 file1.f file2.f file3.f
Link all the object files into the final application.
xlf95 -qpdf2 -O3 file1.o file2.o file3.o file4.o

```

Here is an example that bypasses recompiling the source with `-qpdf2`:

```

Set the PDFDIR variable.
export PDFDIR=$HOME/project_dir
Compile source with -qpdf1.
xlf -O3 -qpdf1 -c file.f
Link in object file.
xlf -O3 -qpdf1 file.o
Run with one set of input data.
a.out <sample.data
Link in object file from qpdf1 pass.
(Bypass source recompilation with -qpdf2.)
xlf -O3 -qpdf2 file.o

```

Here is an example of using `pdf1` and `pdf2` objects:

```

Set the PDFDIR variable.
export PDFDIR=$HOME/project_dir
Compile source with -qpdf1.
xlf -c -qpdf1 -O3 file1.f file2.f
Link in object files.
xlf -qpdf1 -O3 file1.o file2.o
Run with one set of input data.
a.out <sample.data
Link in the mix of pdf1 and pdf2 objects.
xlf -qpdf2 -O3 file1.o file2.o

```

### Related information

- “XL Fortran input files” on page 23
- “XL Fortran Output files” on page 25
- *Benefits of Profile-Directed Feedback in the XL Fortran Optimization and Programming Guide*

## -qphsinfo option

### Syntax

-qphsinfo | -qnophsinfo  
PHSINFO | NOPHSINFO

The **-qphsinfo** compiler option displays timing information on the terminal for each compiler phase.

The output takes the form *number1/number2* for each phase where *number1* represents the CPU time used by the compiler and *number2* represents the total of the compiler time and the time that the CPU spends handling system calls.

### Examples

To compile **app.f**, which consists of 3 compilation units, and report the time taken for each phase of the compilation, enter:

```
xlf90 app.f -qphsinfo
```

The output will look similar to:

```
FORTRAN phase 1 ftphas1 TIME = 0.000 / 0.000
** m_module === End of Compilation 1 ===
FORTRAN phase 1 ftphas1 TIME = 0.000 / 0.000
** testassign === End of Compilation 2 ===
FORTRAN phase 1 ftphas1 TIME = 0.000 / 0.010
** dataassign === End of Compilation 3 ===
HOT - Phase Ends; 0.000/ 0.000
HOT - Phase Ends; 0.000/ 0.000
HOT - Phase Ends; 0.000/ 0.000
W-TRANS - Phase Ends; 0.000/ 0.010
OPTIMIZ - Phase Ends; 0.000/ 0.000
REGALLO - Phase Ends; 0.000/ 0.000
AS - Phase Ends; 0.000/ 0.000
W-TRANS - Phase Ends; 0.000/ 0.000
OPTIMIZ - Phase Ends; 0.000/ 0.000
REGALLO - Phase Ends; 0.000/ 0.000
AS - Phase Ends; 0.000/ 0.000
W-TRANS - Phase Ends; 0.000/ 0.000
OPTIMIZ - Phase Ends; 0.000/ 0.000
REGALLO - Phase Ends; 0.000/ 0.000
AS - Phase Ends; 0.000/ 0.000
1501-510 Compilation successful for file app.f.
```

Each phase is invoked three times, corresponding to each compilation unit. FORTRAN represents front-end parsing and semantic analysis, HOT loop transformations, W-TRANS intermediate language translation, OPTIMIZ high-level optimization, REGALLO register allocation and low-level optimization, and AS final assembly.

Compile **app.f** at the **-O4** optimization level with **-qphsinfo** specified:

```
xlf90 myprogram.f -qphsinfo -O4
```

The following output results:

```
FORTRAN phase 1 ftphas1 TIME = 0.010 / 0.020
** m_module === End of Compilation 1 ===
FORTRAN phase 1 ftphas1 TIME = 0.000 / 0.000
** testassign === End of Compilation 2 ===
FORTRAN phase 1 ftphas1 TIME = 0.000 / 0.000
** dataassign === End of Compilation 3 ===
HOT - Phase Ends; 0.000/ 0.000
HOT - Phase Ends; 0.000/ 0.000
HOT - Phase Ends; 0.000/ 0.000
IPA - Phase Ends; 0.080/ 0.100
1501-510 Compilation successful for file app.f.
IPA - Phase Ends; 0.050/ 0.070
W-TRANS - Phase Ends; 0.010/ 0.030
OPTIMIZ - Phase Ends; 0.020/ 0.020
REGALLO - Phase Ends; 0.040/ 0.040
AS - Phase Ends; 0.000/ 0.000
```

Note that during the IPA (interprocedural analysis) optimization phases, the program has resulted in one compilation unit; that is, all procedures have been inlined.

### Related information

“The compiler phases” on page 303.

## **-qpic option**

### **Syntax**

`-qpic[=suboptions]`

The **-qpic** compiler option generates Position Independent Code (PIC) that can be used in shared libraries.

### **Arguments**

**small | large** The **small** suboption tells the compiler to assume that the size of the Table Of Contents be at most, 64K. The **large** suboption allows the size of the Table Of Contents to be larger than 64K. This suboption allows more addresses to be stored in the Table Of Contents. However, it does generate code that is usually larger than that generated by the **small** suboption.

**-qpic=small** is the default.

Specifying **-qpic=large** has the same effect as passing **-bbigtoc** to the **ld** command.

## -qport option

### Syntax

`-qport[=suboptions]` | `-qnoport`  
`PORT[(suboptions)]` | `NOPORT`

The `-qport` compiler option increases flexibility when porting programs to XL Fortran, providing a number of options to accommodate other Fortran language extensions. A particular suboption will always function independently of other `-qport` and compiler options.

### Arguments

#### `logicals` | `noclogicals`

If you specify this option, the compiler treats all non-zero integers that are used in logical expressions as TRUE. You must specify `-qintlog` for `-qport=logicals` to take effect.

The `-qport=logicals` option is useful when porting applications from other Fortran compilers that expect this behavior. However, it is unsafe to mix programs that use different settings for non-zero integers if they share or pass logical data between them. Data files already written with the default `-qintlog` setting will produce unexpected behavior if read with the `-qport=logicals` option active.

#### `hexint` | `nohexint`

If you specify this option, typeless constant hexadecimal strings are converted to integers when passed as an actual argument to the `INT` intrinsic function. Typeless constant hexadecimal strings not passed as actual arguments to `INT` remain unaffected.

`mod` | `nomod` Specifying this option relaxes existing constraints on the `MOD` intrinsic function, allowing two arguments of the same data type parameter to be of different kind type parameters. The result will be of the same type as the argument, but with the larger kind type parameter value.

#### `nullarg` | `nonnullarg`

For an external or internal procedure reference, specifying this option causes the compiler to treat an empty argument, which is delimited by a left parenthesis and a comma, two commas, or a comma and a right parenthesis, as a null argument. This suboption has no effect if the argument list is empty.

Examples of empty arguments are:

```
call foo(,,z)
call foo(x,,z)
call foo(x,y,)
```

The following program includes a null argument.

#### Fortran program:

```
program nularg
 real(4) res/0.0/
 integer(4) rc
 integer(4), external :: add
 rc = add(%val(2), res, 3.14, 2.18,) ! The last argument is a
```

```

! null argument.
if (rc == 0) then
print *, "res = ", res
else
print *, "number of arguments is invalid."
endif
end program

```

**C program:**

```

int add(int a, float *res, float *b, float *c, float *d)
{
 int ret = 0;
 if (a == 2)
 *res = *b + *c;
 else if (a == 3)
 *res = (*b + *c + *d);
 else
 ret = 1;
 return (ret);
}

```

**sce** | **nosce** By default, the compiler performs short circuit evaluation in selected logical expressions using XL Fortran rules. Specifying **sce** allows the compiler to use non-XL Fortran rules. The compiler will perform short circuit evaluation if the current rules allow it.

**typestmt** | **notypestmt**

The TYPE statement, which behaves in a manner similar to the PRINT statement, is supported whenever this option is specified.

**typlessarg** | **notyplessarg**

Converts all typeless constants to default integers if the constants are actual arguments to an intrinsic procedure whose associated dummy arguments are of integer type. Dummy arguments associated with typeless actual arguments of noninteger type remain unaffected by this option.

Using this option may cause some intrinsic procedures to become mismatched in kinds. Specify **-qxlf77=intarg** to convert the kind to that of the longest argument.

**Related information**

See the section on the **INT** and **MOD** intrinsic functions in the *XL Fortran Language Reference* for further information.

## -qposition option

### Syntax

```
-qposition={appendold | appendunknown} ...
POSITION({APPENDOLD | APPENDUNKNOWN} ...)
```

Positions the file pointer at the end of the file when data is written after an **OPEN** statement with no **POSITION=** specifier and the corresponding **STATUS=** value (**OLD** or **UNKNOWN**) is specified.

The default setting depends on the I/O specifiers in the **OPEN** statement and on the compiler invocation command: **-qposition=appendold** for the **xlF**, **xlF\_r**, **xlF\_r7**, and **f77/fort77** commands and the defined Fortran 90 and Fortran 95 behaviors for the **xlF90**, **xlF90\_r**, **xlF90\_r7**, **xlF95**, **xlF95\_r**, **xlF95\_r7**, **F90** and **F95** commands.

### Rules

The position becomes **APPEND** when the first I/O operation moves the file pointer if that I/O operation is a **WRITE** or **PRINT** statement. If it is a **BACKSPACE**, **ENDFILE**, **READ**, or **REWIND** statement instead, the position is **REWIND**.

### Applicable product levels

The **appendunknown** suboption is the same as the XL Fortran Version 2 **append** suboption, but we recommend using **appendunknown** to avoid ambiguity.

**-qposition=appendold:appendunknown** provides compatibility with XL Fortran Version 1 and early Version 2 behavior. **-qposition=appendold** provides compatibility with XL Fortran Version 2.3 behavior.

### Examples

In the following example, **OPEN** statements that do not specify a **POSITION=** specifier, but specify **STATUS='old'** will open the file as if **POSITION='append'** was specified.

```
xlF95 -qposition=appendold opens_old_files.f
```

In the following example, **OPEN** statements that do not specify a **POSITION=** specifier, but specify **STATUS='unknown'** will open the file as if **POSITION='append'** was specified.

```
xlF95 -qposition=appendunknown opens_unknown_files.f
```

In the following example, **OPEN** statements that do not specify a **POSITION=** specifier, but specify either **STATUS='old'** or **STATUS='unknown'** will open the file as if **POSITION='append'** was specified.

```
xlF95 -qposition=appendold:appendunknown opens_many_files.f
```

### Related information

- *File positioning* in the *XL Fortran Optimization and Programming Guide*
- **OPEN** statement in the *XL Fortran Language Reference*

## **-qprefetch option**

### **Syntax**

**-qprefetch** | -qnoprefetch

Instructs the compiler to insert the prefetch instructions automatically where there are opportunities to improve code performance.

### **Related information**

For more information on prefetch directives, see **PREFETCH directives** in the *XL Fortran Language Reference* and *The POWER4 Processor Introduction and Tuning Guide*. To selectively control prefetch directives using trigger constants, see the “-qdirective option” on page 135.

## **-qqcount option**

### **Syntax**

-qqcount | -qnoqcount  
QCOUNT | NOQCOUNT

Accepts the **Q** character-count edit descriptor (**Q**) as well as the extended-precision **Q** edit descriptor (**Qw.d**). With **-qnoqcount**, all **Q** edit descriptors are interpreted as the extended-precision **Q** edit descriptor.

### **Rules**

The compiler interprets a **Q** edit descriptor as one or the other depending on its syntax and issues a warning if it cannot determine which one is specified.

### **Related information**

See *Q (Character Count) Editing* in the *XL Fortran Language Reference*.

## -qrealsize option

### Syntax

`-qrealsize=bytes`  
`REALSIZE(bytes)`

Sets the default size of **REAL**, **DOUBLE PRECISION**, **COMPLEX**, and **DOUBLE COMPLEX** values.

This option is intended for maintaining compatibility with code that is written for other systems. You may find it useful as an alternative to **-qautodbl** in some situations.

### Rules

The option affects the sizes<sup>2</sup> of constants, variables, derived type components, and functions (which include intrinsic functions) for which no kind type parameter is specified. Objects that are declared with a kind type parameter or length, such as **REAL(4)** or **COMPLEX\*16**, are not affected.

### Arguments

The allowed values for *bytes* are:

- 4 (the default)
- 8

### Results

This option determines the sizes of affected objects as follows:

| Data Object      | REALSIZE(4) in Effect | REALSIZE(8) in Effect |
|------------------|-----------------------|-----------------------|
| 1.2              | REAL(4)               | REAL(8)               |
| 1.2e0            | REAL(4)               | REAL(8)               |
| 1.2d0            | REAL(8)               | REAL(16)              |
| 1.2q0            | REAL(16)              | REAL(16)              |
| REAL             | REAL(4)               | REAL(8)               |
| DOUBLE PRECISION | REAL(8)               | REAL(16)              |
| COMPLEX          | COMPLEX(4)            | COMPLEX(8)            |
| DOUBLE COMPLEX   | COMPLEX(8)            | COMPLEX(16)           |

Similar rules apply to intrinsic functions:

- If an intrinsic function has no type declaration, its argument and return types may be changed by the **-qrealsize** setting.
- Any type declaration for an intrinsic function must agree with the default size of the return value.

This option is intended to allow you to port programs unchanged from systems that have different default sizes for data. For example, you might need **-qrealsize=8** for programs that are written for a CRAY computer. The default value of 4 for this option is suitable for programs that are written specifically for many 32-bit computers.

Setting **-qrealsize** to 8 overrides the setting of the **-qdpc** option.

---

2. In Fortran 90/95 terminology, these values are referred to as *kind type parameters*.

## Examples

This example shows how changing the **-qrealsize** setting transforms some typical entities:

```
@PROCESS REALSIZE(8)
 REAL R ! treated as a real(8)
 REAL(8) R8 ! treated as a real(8)
 DOUBLE PRECISION DP ! treated as a real(16)
 DOUBLE COMPLEX DC ! treated as a complex(16)
 COMPLEX(4) C ! treated as a complex(4)
 PRINT *,DSIN(DP) ! treated as qsin(real(16))
! Note: we cannot get dsin(r8) because dsin is being treated as qsin.
END
```

Specifying **-qrealsize=8** affects intrinsic functions, such as **DABS**, as follows:

```
INTRINSIC DABS ! Argument and return type become REAL(16).
DOUBLE PRECISION DABS ! OK, because DOUBLE PRECISION = REAL(16)
 ! with -qrealsize=8 in effect.
REAL(16) DABS ! OK, the declaration agrees with the option setting.
REAL(8) DABS ! The declaration does not agree with the option
 ! setting and is ignored.
```

## Related information

“-qintsize option” on page 167 is a similar option that affects integer and logical objects. “-qautodbl option” on page 121 is related to **-qrealsize**, although you cannot combine the options. When the **-qautodbl** option turns on automatic doubling, padding, or both, the **-qrealsize** option has no effect.

*Type Parameters and Specifiers* in the *XL Fortran Language Reference* discusses kind type parameters.

## -qrecur option

### Syntax

-qrecur | **-qnorecur**  
RECUR | **NORECUR**

Not recommended. Specifies whether external subprograms may be called recursively. For new programs, use the **RECURSIVE** keyword, which provides a standard-conforming way of using recursive procedures. If you specify the **-qrecur** option, the compiler must assume that any procedure could be recursive. Code generation for recursive procedures may be less efficient. Using the **RECURSIVE** keyword allows you to specify exactly which procedures are recursive.

### Examples

! The following RECUR recursive function:

```
@process recur
function factorial (n)
integer factorial
if (n .eq. 0) then
 factorial = 1
else
 factorial = n * factorial (n-1)
end if
end function factorial
```

! can be rewritten to use F90/F95 RECURSIVE/RESULT features:

```
recursive function factorial (n) result (res)
integer res
if (n .eq. 0) then
 res = 1
else
 res = n * factorial (n-1)
end if
end function factorial
```

### Restrictions

If you use the **xlf**, **xlf\_r**, **xlf\_r7**, **f77**, or **fort77** command to compile programs that contain recursive calls, specify **-qnosave** to make the default storage class automatic.

## -qreport option

### Syntax

```
-qreport[={smplist | hotlist}...]
-qnoreport
REPORT[({SMPLIST | HOTLIST}...)] NOREPORT
```

Determines whether to produce transformation reports showing how the program is parallelized and how loops are optimized.

You can use the **smp`list`** suboption to debug or tune the performance of SMP programs by examining the low-level transformations. You can see how the program deals with data and the automatic parallelization of loops. Comments within the listing tell you how the transformed program corresponds to the original source code and include information as to why certain loops were not parallelized.

You can use the **hot`list`** suboption to generate a report showing how loops are transformed.

### Arguments

#### smp`list`

Produces a pseudo-Fortran listing that shows how the program is parallelized. This listing is produced before loop and other optimizations are performed. It includes messages that point out places in the program that can be modified to be more efficient. This report will only be produced if the **-qsmp** option is in effect.

**hot`list`** Produces a pseudo-Fortran listing that shows how loops are transformed, to assist you in tuning the performance of all loops. This suboption is the default if you specify **-qreport** with no suboptions.

In addition, if you specify the **-qreport=hot`list`** option when the **-qsmp** option is in effect, a pseudo-Fortran listing will be produced that shows the calls to the SMP runtime and the procedures created for parallel constructs.

### Background information

The transformation listing is part of the compiler listing file.

### Restrictions

Loop transformation and auto parallelization are done on the link step at a **-O5** (or **-qipa=level=2**) optimization level. The **-qreport** option will generate the report in the listing file on the link step.

You must specify the **-qsmp** or the **-qhot** option to generate a loop transformation listing. You must specify the **-qsmp** option to generate a parallel transformation listing or parallel performance messages.

The code that the listing shows is not intended to be compilable. Do not include any of this code in your own programs or explicitly call any of the internal routines whose names appear in the listing.

### Examples

To produce a listing file that you can use to tune parallelization:

```
xlf_r -qsmp -O3 -qhot -qreport=smplist needs_tuning.f
```

To produce a listing file that you can use to tune both parallelization and loop performance:

```
xlf_r -qsmp -03 -qhot -qreport=smp1ist:hot1ist needs_tuning.f
```

To produce a listing file that you can use to tune only the performance of loops:

```
xlf95_r -03 -qhot -qreport=hot1ist needs_tuning.f
```

### **Related information**

See “-qpdf option” on page 198.

## **-qsaa option**

### **Syntax**

-qsaa | -qnosaa  
SAA | NOsAA

Checks for conformance to the SAA FORTRAN language definition. It identifies nonconforming source code and also options that allow such nonconformances.

### **Rules**

These warnings have a prefix of **(L)**, indicating a problem with the language level.

### **Restrictions**

The **-qflag** option can override this option.

### **Related information**

Use the “-qlanglvl option” on page 177 to check your code for conformance to international standards.

## -qsave option

### Syntax

```
-qsave[={all|defaultinit}] | -qnosave
SAVE[({all|defaultinit})] NOSAVE
```

This specifies the default storage class for local variables.

If **-qsave=all** is specified, the default storage class is **STATIC**; if **-qnosave** is specified, the default storage class is **AUTOMATIC**; if **-qsave=defaultinit** is specified, the default storage class is **STATIC** for variables of derived type that have default initialization specified, and **AUTOMATIC** otherwise. The default suboption for the **-qsave** option is **all**. The two suboptions are mutually exclusive.

The default for this option depends on the invocation used. For example, you may need to specify **-qsave** to duplicate the behavior of FORTRAN 77 programs. The **xlF**, **xlF\_r**, **xlF\_r7**, **f77**, and **fort77** commands have **-qsave** listed as a default option in **/etc/xlF.cfg** to preserve the previous behavior.

The following example illustrates the impact of the **-qsave** option on derived data type:

```
PROGRAM P
 CALL SUB
 CALL SUB
END PROGRAM P

SUBROUTINE SUB
 LOGICAL, SAVE :: FIRST_TIME = .TRUE.
 STRUCTURE /S/
 INTEGER I/17/
 END STRUCTURE
 RECORD /S/ LOCAL_STRUCT
 INTEGER LOCAL_VAR

 IF (FIRST_TIME) THEN
 LOCAL_STRUCT.I = 13
 LOCAL_VAR = 19
 FIRST_TIME = .FALSE.
 ELSE
 ! Prints " 13" if compiled with -qsave or -qsave=all
 ! Prints " 13" if compiled with -qsave=defaultinit
 ! Prints " 17" if compiled with -qnosave
 PRINT *, LOCAL_STRUCT
 ! Prints " 19" if compiled with -qsave or -qsave=all
 ! Value of LOCAL_VAR is undefined otherwise
 PRINT *, LOCAL_VAR
 END IF
END SUBROUTINE SUB
```

### Related information

The **-qnosave** option is usually necessary for multi-threaded applications and subprograms that are compiled with the “**-qrecur** option” on page 212.

See *Storage Classes for Variables* in the *XL Fortran Language Reference* for information on how this option affects the storage class of variables.

## **-qsaveopt option**

### **Syntax**

`-qsaveopt` | `-qnosaveopt`

Saves the command-line options used for compiling a source file, and other information, in the corresponding object file. The compilation must produce an object file for this option to take effect. Only one copy of the command-line options is saved, even though each object may contain multiple compilation units.

To list the options used, issue the **what -s** command on the object file. The following is listed:

```
opt source_type invocation_used compilation_options
```

For example, if the object file is **t.o**, the **what -s t.o** command may produce information similar to the following:

```
opt f /usr/bin/xlf/xlf90 -qlist -qsaveopt t.f
```

where **f** identifies the source used as Fortran, **/usr/bin/xlf/xlf90** shows the invocation command used, and **-qlist -qsaveopt** shows the compilation options.

## **-qsclk option**

### **Syntax**

`-qsclk[={centi | micro}]`

Specifies the resolution that the **SYSTEM\_CLOCK** intrinsic procedure uses in a program. The default is centisecond resolution (`-qsclk=centi`). To use microsecond resolution, specify `-qsclk=micro`.

### **Related information**

See **SYSTEM\_CLOCK** in the *XL Fortran Language Reference* for more information on returning integer data from a real-time clock.

## **-qshowpdf option**

### **Syntax**

`-qshowpdf` | `-qnshowpdf`

Used together with `-qpdf1` to add additional call and block count profiling information to an executable.

When specified together with `-qpdf1`, the compiler inserts additional profiling information into the compiled application to collect call and block counts for all procedures in the application. Running the compiled application will record the call and block counts to the `._pdf` file.

After you run your application with training data, you can retrieve the contents of the `._pdf` file with the `showpdf` utility. This utility is described in “`-qpdf option`” on page 198.

## -qsigtrap option

### Syntax

`-qsigtrap[=trap_handler]`

When you are compiling a file that contains a main program, this option sets up the specified trap handler to catch **SIGTRAP** exceptions. This option enables you to install a handler for **SIGTRAP** signals without calling the **SIGNAL** subprogram in the program.

### Arguments

To enable the `xl__trce` trap handler, specify **-qsigtrap** without a handler name. To use a different trap handler, specify its name with the **-qsigtrap** option.

If you specify a different handler, ensure that the object module that contains it is linked with the program. To show more detailed information in the tracebacks generated by the trap handlers provided by XL Fortran (such as `xl__trce`), specify the **-qlinedebug** or **-g** option.

### Related information

- “XL Fortran run-time exceptions” on page 49 describes the possible causes of exceptions.
- *Detecting and trapping floating-point exceptions* in the *XL Fortran Optimization and Programming Guide* describes a number of methods for dealing with exceptions that result from floating-point computations.
- *Installing an exception handler* in the *XL Fortran Optimization and Programming Guide* lists the exception handlers that XL Fortran supplies.

## **-qsmallstack option**

### **Syntax**

`-qsmallstack[=[no]dynlenonheap]`  
| `-qnosmallstack`

Specifies that the compiler will minimize stack usage where possible.

This compiler option controls two distinct, but related sets of transformations: general small stack transformations and dynamic length variable allocation transformations. These two kinds of transformations can be controlled independently of each other.

The default, `-qnosmallstack`, implies that all suboptions are off.

The `-qsmallstack=dynlenonheap` suboption affects automatic objects that have nonconstant character lengths or a nonconstant array bound (DYNAMIC LENGTH ON HEAP). When specified, those automatic variables are allocated on the heap. When this suboption is not specified, those automatic variables are allocated on the stack.

Using this option may adversely affect program performance; it should be used only for programs that allocate large amounts of data on the stack.

### **Rules**

- `-qsmallstack` with no suboptions enables only the general small stack transformations.
- `-qnosmallstack` only disables the general small stack transformations. To disable `dynlenonheap` transformations, specify `-qsmallstack=nodynlenonheap` as well.
- `-qsmallstack=dynlenonheap` enables the dynamic length variable allocation and general small stack transformations.
- To enable only the `dynlenonheap` transformations, specify `-qsmallstack=dynlenonheap -qnosmallstack`.
- When both `-qsmallstack` and `-qstacktemp` options are used, the `-qstacktemp` setting will be used to allocate applicable temporary variables if its value is non-zero, even if this setting conflicts with that of `-qsmallstack`. The `-qsmallstack` setting will continue to apply transformations not affected by `-qstacktemp`.

### **Related information**

- “`-qstacktemp` option” on page 229

## -qsmp option

### Syntax

-qsmp [=suboptions]

-qnosmp

Indicates that code should be produced for an SMP system. The default is to produce code for a uniprocessor machine. When you specify this option, the compiler recognizes all directives with the trigger constants **SMP\$**, **\$OMP**, and **IBMP** (unless you specify the **omp** suboption).

Only the **xlf\_r**, **xlf\_r7**, **xlf90\_r**, **xlf90\_r7**, **xlf95\_r**, and **xlf95\_r7** invocation commands automatically link in all of the thread-safe components. You can use the **-qsmp** option with the **xlf**, **xlf90**, **xlf95**, **f77**, and **fort77** invocation commands, but you are responsible for linking in the appropriate components. For a description of this, refer to “Linking 32-bit SMP object files using the **ld** command” on page 31. If you use the **-qsmp** option to compile any source file in a program, then you must specify the **-qsmp** option at link time as well, unless you link by using the **ld** command.

### Arguments

auto | **noauto**

This suboption controls automatic parallelization. By default, the compiler will attempt to parallelize explicitly coded **DO** loops as well as those that are generated by the compiler for array language. If you specify the suboption **noauto**, automatic parallelization is turned off, and only constructs that are marked with prescriptive directives are parallelized. If the compiler encounters the **omp** suboption and the **-qsmp** or **-qsmp=auto** suboptions are not explicitly specified on the command line, the **noauto** suboption is implied. Also, note that **-qsmp=noopt** implies **-qsmp=noauto**. No automatic parallelization occurs under **-qsmp=noopt**; only user-directed parallelization will occur.

nested\_par | nonested\_par

If you specify the **nested\_par** suboption, the compiler parallelizes prescriptive nested parallel constructs (**PARALLEL DO**, **PARALLEL SECTIONS**). This includes not only the loop constructs that are nested within a scoping unit but also parallel constructs in subprograms that are referenced (directly or indirectly) from within other parallel constructs. By default, the compiler serializes a nested parallel construct. Note that this option has no effect on loops that are automatically parallelized. In this case, at most one loop in a loop nest (in a scoping unit) will be parallelized.

Note that the implementation of the **nested\_par** suboption does not comply with the OpenMP API. If you specify this suboption, the run-time library uses the same threads for the nested **PARALLEL DO** and **PARALLEL SECTIONS** constructs that it used for the enclosing **PARALLEL** constructs.

**omp** | noomp

If you specify **-qsmp=omp**, the compiler enforces compliance with the OpenMP API. Specifying this option has the following effects:

- Automatic parallelization is turned off.
- All previously recognized directive triggers are ignored.

- The **-qcclines** compiler option is turned on if you specify **-qsmp=omp**.
- The **-qcclines** compiler option is not turned on if you specify **-qnocclines** and **-qsmp=omp**.
- The only recognized directive trigger is **\$OMP**. However, you can specify additional triggers on subsequent **-qdirective** options.
- The compiler issues warning messages if your code contains any language constructs that do not conform to the OpenMP API.

Specifying this option when the C preprocessor is invoked also defines the **\_OPENMP** C preprocessor macro automatically, with the value *200505*, which is useful in supporting conditional compilation. This macro is only defined when the C preprocessor is invoked.

See *Conditional Compilation* in the *Language Elements* section of the *XL Fortran Language Reference* for more information.

#### **opt | noopt**

If the **-qsmp=noopt** suboption is specified, the compiler will do the smallest amount of optimization that is required to parallelize the code. This is useful for debugging because **-qsmp** enables the **-O2** and **-qhot** options by default, which may result in the movement of some variables into registers that are inaccessible to the debugger. However, if the **-qsmp=noopt** and **-g** options are specified, these variables will remain visible to the debugger.

#### **rec\_locks | norec\_locks**

This suboption specifies whether recursive locks are used to avoid problems associated with **CRITICAL** constructs. If you specify the **rec\_locks** suboption, a thread can enter a **CRITICAL** construct from within the dynamic extent of another **CRITICAL** construct that has the same name. If you specify **norec\_locks**, a deadlock would occur in such a situation.

The default is **norec\_locks**, or regular locks.

#### **schedule=option**

The **schedule** suboption can take any one of the following subsuboptions:

##### **affinity[=*n*]**

The iterations of a loop are initially divided into *number\_of\_threads* partitions, containing **CEILING**(*number\_of\_iterations* / *number\_of\_threads*) iterations. Each partition is initially assigned to a thread and is then further subdivided into chunks that each contain *n* iterations. If *n* has not been specified, then the chunks consist of **CEILING**(*number\_of\_iterations\_left\_in\_partition* / 2) loop iterations.

When a thread becomes free, it takes the next chunk from its initially assigned partition. If there are no more chunks in that partition, then the thread takes the next available chunk from a partition initially assigned to another thread.

The work in a partition initially assigned to a sleeping thread will be completed by threads that are active.

**dynamic[=*n*]**

The iterations of a loop are divided into chunks containing *n* iterations each. If *n* has not been specified, then the chunks consist of **CEILING**(number\_of\_iterations / number\_of\_threads) iterations.

Active threads are assigned these chunks on a "first-come, first-do" basis. Chunks of the remaining work are assigned to available threads until all work has been assigned.

If a thread is asleep, its assigned work will be taken over by an active thread once that thread becomes available.

**guided[=*n*]**

The iterations of a loop are divided into progressively smaller chunks until a minimum chunk size of *n* loop iterations is reached. If *n* has not been specified, the default value for *n* is 1 iteration.

The first chunk contains **CEILING**(number\_of\_iterations / number\_of\_threads) iterations. Subsequent chunks consist of **CEILING**(number\_of\_iterations\_left / number\_of\_threads) iterations. Active threads are assigned chunks on a "first-come, first-do" basis.

**runtime**

Specifies that the chunking algorithm will be determined at run time.

**static[=*n*]**

The iterations of a loop are divided into chunks containing *n* iterations each. Each thread is assigned chunks in a "round-robin" fashion. This is known as *block cyclic scheduling*. If the value of *n* is 1, then the scheduling type is specifically referred to as *cyclic scheduling*.

If you have not specified *n*, the chunks will contain **CEILING**(number\_of\_iterations / number\_of\_threads) iterations. Each thread is assigned one of these chunks. This is known as *block scheduling*.

If a thread is asleep and it has been assigned work, it will be awakened so that it may complete its work.

For more information on chunking algorithms and **SCHEDULE**, refer to the *Directives* section in the *XL Fortran Language Reference*.

**threshold=*n***

Controls the amount of automatic loop parallelization that occurs. The value of *n* represents the lower limit allowed for parallelization of a loop, based on the level of "work" present in a loop. Currently, the calculation of "work" is weighted heavily by the number of iterations in the loop. In general, the higher the value specified for *n*, the fewer loops are parallelized. If this suboption is not specified, the program will use the default value *n*=100.

**Rules**

- If you specify **-qsmp** more than once, the previous settings of all suboptions are preserved, unless overridden by the subsequent suboption setting. The compiler does not override previous suboptions that you specify. The same is true for the version of **-qsmp** without suboptions; the default options are saved.

- Specifying the **omp** suboption always implies **noauto**. Specify **-qsmp=omp:auto** to apply automatic parallelization on OpenMP compliant applications, as well.
- Specifying the **noomp** suboption always implies **auto**.
- The **omp** and **noomp** suboptions only appear in the compiler listing if you explicitly set them.
- If **-qsmp** is specified without any suboptions, **-qsmp=opt** becomes the default setting. If **-qsmp** is specified after the **-qsmp=noopt** suboption has been set, the **-qsmp=noopt** setting will always be ignored.
- If the option **-qsmp** with no suboptions follows the suboption **-qsmp=noopt** on a command line, the **-qsmp=opt** and **-qsmp=auto** options are enabled.
- Specifying the **-qsmp=noopt** suboption implies **-qsmp=noauto**. It also implies **-qnoopt**. This option overrides performance options such as **-O2**, **-O3**, **-qhot**, anywhere on the command line unless **-qsmp** appears after **-qsmp=noopt**.
- Specifying **-qsmp** implies **-qhot** is in effect.
- Object files generated with the **-qsmp=opt** option can be linked with object files generated with **-qsmp=noopt**. The visibility within the debugger of the variables in each object file will not be affected by linking.

## Restrictions

The **-qsmp=noopt** suboption may affect the performance of the program.

Within the same **-qsmp** specification, you cannot specify the **omp** suboption before or after certain suboptions. The compiler issues warning messages if you attempt to specify them with **omp**:

**auto** This suboption controls automatic parallelization, but **omp** turns off automatic parallelization.

### nested\_par

Note that the implementation of the **nested\_par** suboption does not comply with the OpenMP API. If you specify this suboption, the run-time library uses the same threads for the nested **PARALLEL DO** and **PARALLEL SECTIONS** constructs that it used for the enclosing **PARALLEL** constructs.

### rec\_locks

This suboption specifies a behavior for **CRITICAL** constructs that is inconsistent with the OpenMP API.

### schedule=affinity=*n*

The affinity scheduling type does not appear in the OpenMP API standard.

## Examples

The **-qsmp=noopt** suboption overrides performance optimization options anywhere on the command line unless **-qsmp** appears after **-qsmp=noopt**. The following examples illustrate that all optimization options that appear after **-qsmp=noopt** are processed according to the normal rules of scope and precedence.

### Example 1

```
xlf90 -qsmp=noopt -O3...
is equivalent to
xlf90 -qsmp=noopt...
```

### Example 2

```
 xlf90 -qsmp=noopt -O3 -qsmp...
is equivalent to
 xlf90 -qsmp -O3...
```

### Example 3

```
 xlf90 -qsmp=noopt -O3 -qhot -qsmp -O2...
is equivalent to
 xlf90 -qsmp -qhot -O2...
```

If you specify the following, the compiler recognizes both the **\$OMP** and **SMP\$** directive triggers and issues a warning if a directive specified with either trigger is not allowed in OpenMP.

```
-qsmp=omp -qdirective=SMP$
```

If you specify the following, the **noauto** suboption is used. The compiler issues a warning message and ignores the **auto** suboption.

```
-qsmp=omp:auto
```

In the following example, you should specify **-qsmp=rec\_locks** to avoid a deadlock caused by **CRITICAL** constructs.

```
program t
 integer i, a, b

 a = 0
 b = 0
!smp$ parallel do
 do i=1, 10
!smp$ critical
 a = a + 1
!smp$ critical
 b = b + 1
!smp$ end critical
!smp$ end critical
 enddo
end
```

### Related information

If you use the **xlf**, **xlf\_r**, **xlf\_r7**, **f77**, or **fort77** command with the **-qsmp** option to compile programs, specify **-qnosave** to make the default storage class automatic, and specify **-qthreaded** to tell the compiler to generate thread-safe code.

## -qsource option

### Syntax

```
-qsource | -qnosource
SOURCE | NOSOURCE
```

Determines whether to produce the source section of the listing.

This option displays on the terminal each source line where the compiler detects a problem, which can be very useful in diagnosing program errors in the Fortran source files.

You can selectively print parts of the source code by using **SOURCE** and **NOSOURCE** in **@PROCESS** directives in the source files around those portions of the program you want to print. This is the only situation where the **@PROCESS** directive does not have to be before the first statement of a compilation unit.

### Examples

In the following example, the point at which the incorrect call is made is identified more clearly when the program is compiled with the **-qsource** option:

```
$ cat argument_mismatch.f
 subroutine mult(x,y)
 integer x,y
 print *,x*y
 end

 program wrong_args
 interface
 subroutine mult(a,b) ! Specify the interface for this
 integer a,b ! subroutine so that calls to it
 end subroutine mult ! can be checked.
 end interface
 real i,j
 i = 5.0
 j = 6.0
 call mult(i,j)
 end

$ xlf95 argument_mismatch.f
** mult === End of Compilation 1 ===
"argument_mismatch.f", line 16.12: 1513-061 (S) Actual argument attributes
do not match those specified by an accessible explicit interface.
** wrong_args === End of Compilation 2 ===
1501-511 Compilation failed for file argument_mismatch.f.
$ xlf95 -qsource argument_mismatch.f
** mult === End of Compilation 1 ===
 16 | call mult(i,j)
 a...
a - 1513-061 (S) Actual argument attributes do not match those specified by
an accessible explicit interface.
** wrong_args === End of Compilation 2 ===
1501-511 Compilation failed for file argument_mismatch.f.
```

### Related information

See "Options that control listings and messages" on page 62 and "Source section" on page 296.

## **-qspillsize option**

### **Syntax**

`-qspillsize=bytes`  
`SPILLSIZE(bytes)`

`-qspillsize` is the long form of `-NS`. See “`-NS` option” on page 99.

## -qstacktemp option

### Syntax

`-qstacktemp={0 | -1 | positive signed integer value}`  
`STACKTEMP={0 | -1 | positive signed integer value}`

Determines where to allocate applicable XL Fortran compiler temporaries at run time.

Applicable compiler temporaries are the set of temporary variables created by the compiler for its own use when it determines it can safely apply these. Most typically, the compiler creates these kinds of temporaries to hold copies of XL Fortran arrays for array language semantics, especially in conjunction with calls to intrinsic functions or user subprograms. If you have programs that make use of large arrays, you may need to use this option to help prevent stack space overflow when running them. For example, for SMP or OpenMP applications that are constrained by stack space, you can use this option to move some compiler temporaries onto the heap from the stack.

The compiler cannot detect whether or not the stack limits will be exceeded when an application runs. You will need to experiment with several settings before finding the one that works for your application. To override an existing setting, you must specify a new setting.

**Note:** The `-qstacktemp` option can take precedence over the `-qsmallstack` option for certain compiler-generated temporaries.

### Arguments

The possible suboptions are:

- 0** Based on the target environment, the compiler determines whether it will allocate applicable temporaries on the heap or the stack. If this setting causes your program to run out of stack storage, try specifying a nonzero value instead, or try using the `-qsmallstack` option.
- 1** Allocates applicable temporaries on the stack. Generally, this is the best performing setting but uses the most amount of stack storage.

*positive signed integer value*

Allocates applicable temporaries less than this *value* on the stack and those greater than or equal to this *value* on the heap. A value of 1 Mb has been shown to be a good compromise between stack storage and performance for many programs, but you may need to adjust this number based on your application's characteristics.

## -qstrict option

### Syntax

-qstrict | -qnostrict  
STRICT | NOSTRICT

Ensures that optimizations done by default with the **-O3**, **-O4**, and **-O5** options, and optionally with the **-O2** option, do not alter the semantics of a program.

### Defaults

For **-O3**, **-O4**, and **-O5**, the default is **-qnostrict**. For **-O2**, the default is **-qstrict**. This option is ignored for **-qnoopt**. With **-qnostrict**, optimizations may rearrange code so that results or exceptions are different from those of unoptimized programs.

This option is intended for situations where the changes in program execution in optimized programs produce different results from unoptimized programs. Such situations are likely rare because they involve relatively little-used rules for IEEE floating-point arithmetic.

### Rules

With **-qnostrict** in effect, the following optimizations are turned on, unless **-qstrict** is also specified:

- Code that may cause an exception may be rearranged. The corresponding exception might happen at a different point in execution or might not occur at all. (The compiler still tries to minimize such situations.)
- Floating-point operations may not preserve the sign of a zero value. (To make certain that this sign is preserved, you also need to specify **-qfloat=rrm**, **-qfloat=nomaf**, or **-qfloat=strictnmaf**.)
- Floating-point expressions may be reassociated. For example,  $(2.0*3.1)*4.2$  might become  $2.0*(3.1*4.2)$  if that is faster, even though the result might not be identical.
- The **fltint** and **rsqrt** suboptions of the **-qfloat** option are turned on. You can turn them off again by also using the **-qstrict** option or the **nofltint** and **norsqrt** suboptions of **-qfloat**. With lower-level or no optimization specified, these suboptions are turned off by default.

### Related information

See “-O option” on page 100, “-qhot option” on page 159, and “-qfloat option” on page 152.

## -qstrictieemod option

### Syntax

`-qstrictieemod` | `-qnostrictieemod`  
`STRICTIEEMOD` | `NOSTRICTIEEMOD`

Specifies whether the compiler will adhere to the Fortran 2003 IEEE arithmetic rules for the `ieee_arithmetic` and `ieee_exceptions` intrinsic modules. When you specify `-qstrictieemod`, the compiler adheres to the following rules:

- If there is an exception flag set on entry into a procedure that uses the IEEE intrinsic modules, the flag is set on exit. If a flag is clear on entry into a procedure that uses the IEEE intrinsic modules, the flag can be set on exit.
- If there is an exception flag set on entry into a procedure that uses the IEEE intrinsic modules, the flag clears on entry into the procedure and resets when returning from the procedure.
- When returning from a procedure that uses the IEEE intrinsic modules, the settings for halting mode and rounding mode return to the values they had at procedure entry.
- Calls to procedures that do not use the `ieee_arithmetic` or `ieee_exceptions` intrinsic modules from procedures that do use these modules, will not change the floating-point status except by setting exception flags.

Since the above rules can impact performance, specifying `-qnostrictieemod` will relax the rules on saving and restoring floating-point status. This prevents any associated impact on performance.

## -qstrict\_induction option

### Syntax

-qSTRICT\_INDUCtion | -qNOSTRICT\_INDUCtion

Prevents the compiler from performing induction (loop counter) variable optimizations. These optimizations may be *unsafe* (may alter the semantics of your program) when there are integer overflow operations involving the induction variables.

You should avoid specifying **-qstrict\_induction** unless absolutely necessary, as it may cause performance degradation.

### Examples

Consider the following two examples:

#### Example 1

```
integer(1) :: i, j ! Variable i can hold a
j = 0 ! maximum value of 127.

do i = 1, 200 ! Integer overflow occurs when 128th
 j = j + 1 ! iteration of loop is attempted.
enddo
```

#### Example 2

```
integer(1) :: i ! Variable i can hold a maximum
i = 1_1 ! value of 127.

100 continue
 if (i == -127) goto 200 ! Go to label 200 once decimal overflow
 i = i + 1_1 ! occurs and i == -127.
 goto 100
200 continue
 print *, i
end
```

If you compile these examples with the **-qstrict\_induction** option, the compiler does not perform induction variable optimizations, but the performance of the code may be affected. If you compile the examples with the **-qnostrict\_induction** option, the compiler may perform optimizations that may alter the semantics of the programs.

## -qsuffix option

### Syntax

`-qsuffix=option=suffix`

Specifies the source-file suffix on the command line instead of in the `xlfcfg` file. This option saves time for the user by permitting files to be used as named with minimal makefile modifications and removes the risk of problems associated with modifying the `xlfcfg` file. Only one setting is supported at any one time for any particular file type.

### Arguments

`f=suffix`

Where *suffix* represents the new *source-file-suffix*

`o=suffix`

Where *suffix* represents the new *object-file-suffix*

`s=suffix`

Where *suffix* represents the new *assembler-source-file-suffix*

`cpp=suffix`

Where *suffix* represents the new *preprocessor-source-file-suffix*

### Rules

- The new suffix setting is case-sensitive.
- The new suffix can be of any length.
- Any setting for a new suffix will override the corresponding default setting in the `xlfcfg` file.
- If both `-qsuffix` and `-F` are specified, `-qsuffix` is processed last, so its setting will override the setting in the `xlfcfg` file.

### Examples

For instance,

```
xlfc a1.f2k a2.F2K -qsuffix=f=f2k:cpp=F2K
```

will cause these effects:

- The compiler is invoked for source files with a suffix of `.f2k` and `.F2K`.
- `cpp` is invoked for files with a suffix of `.F2K`.

## -qsuppress option

### Syntax

-qsuppress [=nnnn-mmm[:nnnn-mmm ...] | cmpmsg]

-qnosuppress

### Arguments

*nnnn-mmm[:nnnn-mmm ...]*

Suppresses the display of a specific compiler message (*nnnn-mmm*) or a list of messages (*nnnn-mmm[:nnnn-mmm ...]*). *nnnn-mmm* is the message number. To suppress a list of messages, separate each message number with a colon.

### cmpmsg

Suppresses the informational messages that report compilation progress and a successful completion.

This sub-option has no effect on any error messages that are emitted.

### Background information

In some situations, users may receive an overwhelming number of compiler messages. In many cases, these compiler messages contain important information. However, some messages contain information that is either redundant or can be safely ignored. When multiple error or warning messages appear during compilation, it can be very difficult to distinguish which messages should be noted. By using **-qsuppress**, you can eliminate messages that do not interest you.

- The compiler tracks the message numbers specified with **-qsuppress**. If the compiler subsequently generates one of those messages, it will not be displayed or entered into the listing.
- Only compiler and driver messages can be suppressed. Linker or operating system message numbers will be ignored if specified on the **-qextname** compiler option.
- If you are also specifying the **-qipa** compiler option, then **-qipa** must appear before the **-qextname** compiler option on the command line for IPA messages to be suppressed.

### Restrictions

- The value of *nnnn* must be a four-digit integer between 1500 and 1585, since this is the range of XL Fortran message numbers.
- The value of *mmm* must be any three-digit integer (with leading zeros if necessary).

## Examples

```
@process nullterm
 i = 1; j = 2;
 call printf("i=%d\n",%val(i));
 call printf("i=%d, j=%d\n",%val(i),%val(j));
end
```

Compiling this sample program would normally result in the following output:

```
"t.f", line 4.36: 1513-029 (W) The number of arguments to "printf" differ
from the number of arguments in a previous reference. You should use the
OPTIONAL attribute and an explicit interface to define a procedure with
optional arguments.
** _main === End of Compilation 1 ===
1501-510 Compilation successful for file t.f.
```

When the program is compiled with **-qsuppress=1513-029**, the output is:

```
** _main === End of Compilation 1 ===
1501-510 Compilation successful for file t.f.
```

## Related information

For another type of message suppression, see “-qflag option” on page 151.

## -qswapomp option

### Syntax

```
-qswapomp | -qnoswapomp
SWAPOMP | NOSWAPOMP
```

Specifies that the compiler should recognize and substitute OpenMP routines in XL Fortran programs.

The OpenMP routines for Fortran and C have different interfaces. To support multi-language applications that use OpenMP routines, the compiler needs to recognize OpenMP routine names and substitute them with the XL Fortran versions of these routines, regardless of the existence of other implementations of such routines.

The compiler does not perform substitution of OpenMP routines when you specify the **-qnoswapomp** option.

### Restrictions

The **-qswapomp** and **-qnoswapomp** options only affect Fortran sub-programs that reference OpenMP routines that exist in the program.

### Rules

- If a call to an OpenMP routine resolves to a dummy procedure, module procedure, an internal procedure, a direct invocation of a procedure itself, or a statement function, the compiler will not perform the substitution.
- When you specify an OpenMP routine, the compiler substitutes the call to a different special routine depending upon the setting of the **-qintsize** option. In this manner, OpenMP routines are treated as generic intrinsic procedures.
- Unlike generic intrinsic procedures, if you specify an OpenMP routine in an **EXTERNAL** statement, the compiler will not treat the name as a user-defined external procedure. Instead, the compiler will still substitute the call to a special routine depending upon the setting of the **-qintsize** option.
- An OpenMP routine cannot be extended or redefined, unlike generic intrinsic procedures.

### Examples

In the following example, the OpenMP routines are declared in an **INTERFACE** statement.

```
@PROCESS SWAPOMP

INTERFACE
 FUNCTION OMP_GET_THREAD_NUM()
 INTEGER OMP_GET_THREAD_NUM
 END FUNCTION OMP_GET_THREAD_NUM

 FUNCTION OMP_GET_NUM_THREADS()
 INTEGER OMP_GET_NUM_THREADS
 END FUNCTION OMP_GET_NUM_THREADS
END INTERFACE

IAM = OMP_GET_THREAD_NUM()
NP = OMP_GET_NUM_THREADS()
PRINT *, IAM, NP
END
```

**Related information**

See the *OpenMP Execution Environment Routines and Lock Routines* section in the *XL Fortran Language Reference*.

## **-qtbtable option**

### **Syntax**

`-qtbtable={none | small | full}`

Limits the amount of debugging traceback information in object files, which reduces the size of the program.

You can use this option to make your program smaller, at the cost of making it harder to debug. When you reach the production stage and want to produce a program that is as compact as possible, you can specify **-qtbtable=none**. Otherwise, the usual defaults apply: code compiled with **-g** or without **-O** has full traceback information (**-qtbtable=full**), and code compiled with **-O** contains less (**-qtbtable=small**).

### **Arguments**

- none** The object code contains no traceback information at all. You cannot debug the program, because a debugger or other code-examination tool cannot unwind the program's stack at run time. If the program stops because of a run-time exception, it does not explain where the exception occurred.
- small** The object code contains traceback information but not the names of procedures or information about procedure parameters. You can debug the program, but some non-essential information is unavailable to the debugger. If the program stops because of a run-time exception, it explains where the exception occurred but reports machine addresses rather than procedure names.
- full** The object code contains full traceback information. The program is debuggable, and if it stops because of a run-time exception, it produces a traceback listing that includes the names of all procedures in the call chain.

### **Background information**

This option is most suitable for programs that contain many long procedure names, such as the internal names constructed for module procedures. You may find it more applicable to C++ programs than to Fortran programs.

### **Restrictions**

To use the performance tools, such as **tprof**, in the AIX Performance Toolbox, you must compile the Fortran programs with **-qtbtable=full**.

### **Related information**

- “-g option” on page 94
- “-qcompact option” on page 129
- “-O option” on page 100
- *Debugging optimized code* in the *XL Fortran Optimization and Programming Guide*

## **-qthreaded option**

### **Syntax**

`-qthreaded`

Used by the compiler to determine when it must generate thread-safe code.

The **-qthreaded** option does not imply the **-qnosave** option. The **-qnosave** option specifies a default storage class of automatic for user local variables. In general, both of these options need to be used to generate thread-safe code. Specifying these options ensures that variables and code created by the compiler are threadsafe; it does not guarantee the thread safety of user-written code. If you use the **ENTRY** statement to have an alternate entry point for a subprogram and the **xlf\_r** command to compile, also specify the **-qxlf77=nopersistent** option to be thread-safe. You should implement the appropriate locking mechanisms, as well.

### **Defaults**

**-qthreaded** is the default for the **xlf90\_r**, **xlf90\_r7**, **xlf95\_r**, **xlf95\_r7**, **xlf\_r**, and **xlf\_r7** commands.

Specifying the **-qthreaded** option implies **-qdirective=ibmt**, and by default, the *trigger\_constant* **IBMT** is recognized.

## -qtune option

### Syntax

`-qtune=implementation`

Tunes instruction selection, scheduling, and other implementation-dependent performance enhancements for a specific implementation of a hardware architecture. The compiler will use a **-qtune** setting that is compatible with the target architecture, which is controlled by the **-qarch**, **-q32**, and **-q64** options.

If you want your program to run on more than one architecture, but to be tuned to a particular architecture, you can use a combination of the **-qarch** and **-qtune** options. These options are primarily of benefit for floating-point intensive programs.

By arranging (scheduling) the generated machine instructions to take maximum advantage of hardware features such as cache size and pipelining, **-qtune** can improve performance. It only has an effect when used in combination with options that enable optimization.

Although changing the **-qtune** setting may affect the performance of the resulting executable, it has no effect on whether the executable can be executed correctly on a particular hardware platform.

### Arguments

|               |                                                                                                                                                                                                                                                                                                                                                                 |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>auto</b>   | Automatically detects the specific processor type of the compiling machine. It assumes that the execution environment will be the same as the compilation environment.                                                                                                                                                                                          |
| <b>pwr</b>    | The optimizations are tuned for the POWER processors.                                                                                                                                                                                                                                                                                                           |
| <b>pwr2</b>   | The optimizations are tuned for the POWER2 processors.<br><b>pwrx</b> is a synonym for <b>pwr2</b> , but <b>pwr2</b> is preferred.                                                                                                                                                                                                                              |
| <b>pwr2s</b>  | The optimizations are tuned for the desktop implementation of the POWER2 architecture, which has a narrower processor-to-memory bus than other POWER2 implementations. Quad-word instructions, which are slower on these machines than on other POWER2 machines, are deemphasized to reduce bus contention. That is, there may be fewer of them or none at all. |
| <b>p2sc</b>   | The optimizations are tuned for the POWER2 Super Chip.                                                                                                                                                                                                                                                                                                          |
| <b>601</b>    | The optimizations are tuned for the PowerPC 601 processor.                                                                                                                                                                                                                                                                                                      |
| <b>603</b>    | The optimizations are tuned for the PowerPC 603 processor.                                                                                                                                                                                                                                                                                                      |
| <b>604</b>    | The optimizations are tuned for the PowerPC 604 processor.                                                                                                                                                                                                                                                                                                      |
| <b>rs64a</b>  | The optimizations are tuned for the RS64I processor.                                                                                                                                                                                                                                                                                                            |
| <b>rs64b</b>  | The optimizations are tuned for the RS64II processor.                                                                                                                                                                                                                                                                                                           |
| <b>rs64c</b>  | The optimizations are tuned for the RS64III processor.                                                                                                                                                                                                                                                                                                          |
| <b>pwr3</b>   | The optimizations are tuned for the POWER3 processors.                                                                                                                                                                                                                                                                                                          |
| <b>pwr4</b>   | The optimizations are tuned for the POWER4 processors.                                                                                                                                                                                                                                                                                                          |
| <b>pwr5</b>   | The optimizations are tuned for the POWER5 processors.                                                                                                                                                                                                                                                                                                          |
| <b>ppc970</b> | The optimizations are tuned for the PowerPC 970 processors.                                                                                                                                                                                                                                                                                                     |

If you do not specify **-qtune**, its setting is determined by the setting of the **-qarch** option, as follows:

| <b>-qarch Setting</b>       | <b>Allowed -qtune Settings</b>                                                           | <b>Default -qtune Setting</b> |
|-----------------------------|------------------------------------------------------------------------------------------|-------------------------------|
| com (if you specify -q32)   | pwr, pwr2/pwrx, pwr3, pwr4, pwr2s, p2sc, rs64a, rs64b, rs64c, 601, 603, 604, auto        | pwr3 (if you specify -q32)    |
| com (if you specify -q64)   | See the list of acceptable <b>-qtune</b> settings under the <b>-qarch=ppc64</b> entry.   | pwr3 (if you specify -q64)    |
| pwr                         | pwr, pwr2/pwrx, pwr2s, p2sc, 601, auto                                                   | pwr2                          |
| pwr2/pwrx                   | pwr2/pwrx, p2sc, pwr2s                                                                   | pwr2/pwrx                     |
| pwr2s                       | pwr2s, auto                                                                              | pwr2s                         |
| p2sc                        | p2sc, auto                                                                               | p2sc                          |
| 601                         | 601, auto                                                                                | 601                           |
| 603                         | 603, auto                                                                                | 603                           |
| 604                         | 604, auto                                                                                | 604                           |
| ppc (if you specify -q32)   | 601, 603, 604, rs64a, rs64b, rs64c, pwr3, pwr4, pwr5, ppc970, auto                       | pwr3 (if you specify -q32)    |
| ppc (if you specify -q64)   | See the list of acceptable <b>-qtune</b> settings under the <b>-qarch=ppc64</b> entry.   | pwr3 (if you specify -q64)    |
| ppcgr (if you specify -q32) | 603, 604, rs64b, rs64c, pwr3, pwr4, pwr5, ppc970, auto                                   | pwr3 (if you specify -q32)    |
| ppcgr (if you specify -q64) | See the list of acceptable <b>-qtune</b> settings under the <b>-qarch=ppc64gr</b> entry. | pwr3 (if you specify -q64)    |
| ppc64                       | rs64a, rs64b, rs64c, pwr3, pwr4, pwr5, ppc970, auto                                      | pwr3                          |
| ppc64gr                     | rs64b, rs64c, pwr3, pwr4, pwr5, ppc970, auto                                             | pwr3                          |
| ppc64grsq                   | rs64b, rs64c, pwr3, pwr4, pwr5, ppc970, auto                                             | pwr3                          |
| rs64a                       | rs64a, auto                                                                              | rs64a                         |
| rs64b                       | rs64b, auto                                                                              | rs64b                         |
| rs64c                       | rs64c, auto                                                                              | rs64c                         |
| pwr3                        | pwr3, pwr4, pwr5, ppc970, auto                                                           | pwr3                          |
| pwr4                        | pwr4, pwr5, ppc970, auto                                                                 | pwr4                          |
| pwr5                        | pwr5, auto                                                                               | pwr5                          |
| pwr5x                       | pwr5, auto                                                                               | pwr5                          |
| ppc970                      | ppc970, auto                                                                             | ppc970                        |

Note that you can specify any **-qtune** suboption with **-qarch=auto** as long as you are compiling on a machine that is compatible with the **-qtune** suboption. For example, if you specify **-qarch=auto** and **-qtune=pwr5**, you must compile on a POWER3, POWER4, or POWER5 machine.

## Restrictions

Because reducing quad-word instructions may degrade performance on other POWER2 models, we do not recommend the **pwr2s** suboption for programs that

will be run on a number of different POWER2 models. If the program will be run on a set of different POWER2 models, leave the **-qtune** setting as **pwr2**.

### **Related information**

See “-qarch option” on page 114, “-qcache option” on page 124, and “Compiling for specific architectures” on page 28.

## **-qundef option**

### **Syntax**

`-qundef` | `-qnundef`  
`UNDEF` | `NOUNDEF`

`-qundef` is the long form of the “-u option” on page 263.

## -qunroll option

### Syntax

`-qunroll[=auto | yes] | -qnounroll`

Specifies whether unrolling **DO** loops is allowed in a program. Unrolling is allowed on outer and inner **DO** loops.

### Arguments

- auto** The compiler performs basic loop unrolling. This is the default if **-qunroll** is not specified on the command line.
- yes** The compiler looks for more opportunities to perform loop unrolling than that performed with **-qunroll=auto**. Specifying **-qunroll** with no suboptions is equivalent to **-qunroll=yes**. In general, this suboption has more chances to increase compile time or program size than **-qunroll=auto** processing, but it may also improve your application's performance.

If you decide to unroll a loop, specifying one of the above suboptions does not automatically guarantee that the compiler will perform the operation. Based on the performance benefit, the compiler will determine whether unrolling will be beneficial to the program. Experienced compiler users should be able to determine the benefit in advance.

### Rules

The **-qnounroll** option prohibits unrolling unless you specify the **STREAM\_UNROLL**, **UNROLL**, or **UNROLL\_AND\_FUSE** directive for a particular loop. These directives always override the command line options.

### Examples

In the following example, the **UNROLL(2)** directive is used to tell the compiler that the body of the loop can be replicated so that the work of two iterations is performed in a single iteration. Instead of performing 1000 iterations, if the compiler unrolls the loop, it will only perform 500 iterations.

```
!IBM* UNROLL(2)
DO I = 1, 1000
 A(I) = I
END DO
```

If the compiler chooses to unroll the previous loop, the compiler translates the loop so that it is essentially equivalent to the following:

```
DO I = 1, 1000, 2
 A(I) = I
 A(I+1) = I + 1
END DO
```

### Related information

See the appropriate directive on unrolling loops in the *XL Fortran Language Reference*:

- **STREAM\_UNROLL**
- **UNROLL**
- **UNROLL\_AND\_FUSE**

See *Benefits of High Order Transformation in the XL Fortran Optimization and Programming Guide*.

## **-qunwind option**

### **Syntax**

|                 |  |            |
|-----------------|--|------------|
| <b>-qunwind</b> |  | -qnounwind |
| <b>UNWIND</b>   |  | NOUNWIND   |

Specifies that the compiler will preserve the default behavior for saves and restores to volatile registers during a procedure call. If you specify **-qnounwind**, the compiler rearranges subprograms to minimize saves and restores to volatile registers. This rearrangement may make it impossible for the program or debuggers to walk through or "unwind" subprogram stack frame chains.

While code semantics are preserved, applications such as exception handlers that rely on the default behavior for saves and restores can produce undefined results. When using **-qnounwind** in conjunction with the **-g** compiler option, debug information regarding exception handling when unwinding the program's stack can be inaccurate.

## **-qvecnv ol option**

### **Syntax**

`-qvecnv ol` | `-qnovecnv ol`

Specifies whether to use volatile or non-volatile vector registers. Volatile vector registers are those registers whose value is not preserved across function calls or across save context, jump or switch context system library functions. **-qvecnv ol** instructs the compiler to use both volatile and non-volatile vector registers. **-qnovecnv ol** instructs the compiler to use only volatile vector registers.

This option is required for programs where there is risk of interaction between modules built with AIX libraries prior to AIX 5.3 with 5300-03 and vector register use. Restricting the compiler to use only volatile registers will make your vector programs safe, but it potentially forces the compiler to store vector data to memory more often and thus may reduce your application's performance.

### **Notes**

- This option is supported on AIX 5.3 with 5300-03, or higher.
- To use **-qvecnv ol**, you need `bos.adt.include` version 5.3.0.30 or greater installed on your system.
- If **-qnoenablevmx** is in effect, **-qnovecnv ol** is ignored.
- The **-qnovecnv ol** option performs independently of the **-qhot=[no]simd** option and **NOSIMD** VMX directive.
- On AIX 5.3 with 5300-03, only 20 volatile registers (vr0-vr19) are used by default; the 12 non-volatile vector registers (vr20 - vr31) are not used. You can use these registers only if **-qvecnv ol** is in effect.
- Only enable **-qvecnv ol** for older programs that do not perform saves and restores to non-volatile registers. Using **-qvecnv ol** and linking to such code may result in a run-time failure.

### **Related information**

- “-qenablevmx option” on page 141
- “-qhot option” on page 159

## **-qversion option**

### **Syntax**

`-qversion` | `-qnoversion`

Displays the version and release of the compiler being invoked.

Specify this option on its own with the compiler command. For example:

```
xlf90 -qversion
```

## **-qwarn64 option**

### **Syntax**

`-qwarn64` | `-qnowarn64`

Aids in porting code from a 32-bit environment to a 64-bit environment by detecting the truncation of an 8-byte integer pointer to 4 bytes. The **-qwarn64** option uses informational messages to identify statements that may cause problems with the 32-bit to 64-bit migration.

### **Rules**

- The default setting is **-qnowarn64**.
- You can use the **-qwarn64** option in both 32-bit and 64-bit modes.
- The compiler flags the following situations with informational messages:
  - The assignment of a reference to the **LOC** intrinsic to an **INTEGER(4)** variable.
  - The assignment between an **INTEGER(4)** variable or **INTEGER(4)** constant and an integer pointer.
  - The specification of an integer pointer within a common block. We recommend the **-qextchk** option for common block length changes.
  - The specification of an integer pointer within an equivalence statement.
- We additionally recommend the **-qextchk** option and interface blocks for argument checking.

### **Related information**

- “-q32 option” on page 105
- “-q64 option” on page 106
- Chapter 6, “Using XL Fortran in a 64-Bit Environment,” on page 271

## **-qxflag=dvz option**

### **Syntax**

`-qxflag=dvz`

Specifying **-qxflag=dvz** causes the compiler to generate code to detect floating-point divide-by-zero operations.

This option is only effective when you specify an optimization level of **-O** or higher.

With this option on, the extra code calls the external handler function `__xl_dzx` when the divisor is zero. The return value of this function is used as the result of the division. Users are required to provide the function to handle the divide-by-zero operations. Specifying **-qxflag=dvz** handles only single-precision (REAL\*4) and double-precision (REAL\*8) division.

The interface of the function is as follows:

```
real(8) function __xl_dzx(x, y, kind_type)
 real(8), value :: x, y
 integer, value :: kind_type
end function
```

where:

`x` is the dividend value

`y` is the divisor value

### **kind\_type**

specifies the size of the actual arguments associated with `x` and `y`.

A **kind\_type** value equal to zero indicates that the actual arguments associated with `x` and `y` are of type REAL(8). A **kind\_type** value equal to one indicates that the actual arguments associated with `x` and `y` are of type REAL(4).

The division always executes before the handler routine is called. This means that any exception is posted and handled before the handler function is called.

### **Related information**

- *Implementation details of XL Fortran floating-point processing* in the *XL Fortran Optimization and Programming Guide*
- “-qfltrap option” on page 154
- “Understanding XL Fortran error messages” on page 275

## **-qxflag=oldtab option**

### **Syntax**

`-qxflag=oldtab`  
`XFLAG(OLDTAB)`

Interprets a tab in columns 1 to 5 as a single character (for fixed source form programs), for compatibility with XL Fortran Version 1.

### **Defaults**

By default, the compiler allows 66 significant characters on a source line after column 6. A tab in columns 1 through 5 is interpreted as the appropriate number of blanks to move the column counter past column 6. This default is convenient for those who follow the earlier Fortran practice of including line numbers or other data in columns 73 through 80.

### **Rules**

If you specify the option `-qxflag=oldtab`, the source statement still starts immediately after the tab, but the tab character is treated as a single character for counting columns. This setting allows up to 71 characters of input, depending on where the tab character occurs.

## **-qxflag=xalias option**

### **Syntax**

-qxflag=xalias  
XFLAG(XALIAS)

Obsolete: replaced by **-qalias=nostd**. See “-qalias option” on page 109 instead.

## -qxlf77 option

### Syntax

`-qxlf77=settings`  
`XLf77(settings)`

Provides backward compatibility with XL Fortran Versions 1 and 2 aspects of language semantics and I/O data format that have changed. Most of these changes are required by the Fortran 90 standard.

### Defaults

By default, the compiler uses settings that apply to Fortran 95, Fortran 90, and the most recent compiler version in all cases; the default suboptions are **blankpad**, **nogedit77**, **nointarg**, **nointxor**, **leadzero**, **nooldboz**, **nopersistent**, and **nosofteof**. However, these defaults are only used by the `xlf95`, `xlf95_r`, `xlf95_r7`, `xlf90`, `xlf90_r`, `xlf90_r7`, `f90`, and `f95` commands, which you should use to compile new programs.

For maximum compatibility for programs and data created for XL Fortran Versions 1 and 2, the `xlf`, `xlf_r`, `xlf_r7`, `f77`, and `fort77` commands use the opposite settings for this option.

If you only want to compile and run old programs unchanged, you can continue to use the appropriate invocation command and not concern yourself with this option. You should only use this option if you are using existing source or data files with Fortran 90 or Fortran 95 and the `xlf90`, `xlf90_r`, `xlf90_r7`, `xlf95`, `xlf95_r`, `xlf95_r7`, `f90`, or `f95` command and find some incompatibility because of behavior or data format that has changed since XL Fortran Version 2. Eventually, you should be able to recreate the data files or modify the source files to remove the dependency on the old behavior.

### Arguments

To get various aspects of XL Fortran Version 2 behavior, select the nondefault choice for one or more of the following suboptions. The descriptions explain what happens when you specify the nondefault choices.

#### **blankpad** | **noblankpad**

For internal, direct-access, and stream-access files, uses a default setting equivalent to **pad='no'**. This setting produces conversion errors when reading from such a file if the format requires more characters than the record has, thus duplicating the XL Fortran Version 2 behavior. This suboption does not affect direct-access or stream-access files opened with a **pad=** specifier.

#### **gedit77** | **nogedit77**

Uses FORTRAN 77 semantics for the output of **REAL** objects with the **G** edit descriptor. Between FORTRAN 77 and Fortran 90, the representation of 0 for a list item in a formatted output statement changed, as did the rounding method, leading to different output for some combinations of values and **G** edit descriptors.

#### **intarg** | **nointarg**

Converts all integer arguments of an intrinsic procedure to the kind of the longest argument if they are of different kinds. Under Fortran 90/95 rules, some intrinsics (for example, **IBSET**) determine the result type based on the kind of the first argument; others (for example, **MIN** and **MAX**) require that all arguments be of the same kind.

**intxor** | **nointxor**

Treats **.XOR.** as a logical binary intrinsic operator. It has a precedence equivalent to the **.EQV.** and **.NEQV.** operators and can be extended with an operator interface. (Because the semantics of **.XOR.** are identical to those of **.NEQV.**, **.XOR.** does not appear in the Fortran 90 or Fortran 95 language standard.)

Otherwise, the **.XOR.** operator is only recognized as a defined operator. The intrinsic operation is not accessible, and the precedence depends on whether the operator is used in a unary or binary context.

**leadzero** | **noleadzero**

Produces a leading zero in real output under the **D**, **E**, **L**, **F**, and **Q** edit descriptors.

**oldboz** | **nooldboz**

Turns blanks into zeros for data read by **B**, **O**, and **Z** edit descriptors, regardless of the **BLANK=** specifier or any **BN** or **BZ** control edit descriptors. It also preserves leading zeros and truncation of too-long output, which is not part of the Fortran 90 or Fortran 95 standard.

**persistent** | **nopersistent**

Saves the addresses of arguments to subprograms with **ENTRY** statements in static storage, for compatibility with XL Fortran Version 2. This is an implementation choice that has been changed for increased performance.

**softeof** | **nosofteof**

Allows **READ** and **WRITE** operations when a unit is positioned after its endfile record unless that position is the result of executing an **ENDFILE** statement. This suboption reproduces a FORTRAN 77 extension of earlier versions of XL Fortran that some existing programs rely on.

**Related information**

See “Avoiding or fixing upgrade problems” on page 16.

## -qxlf90 option

### Syntax

-qxlf90={*settings*}  
XLF90({*settings*})

Provides backward compatibility with XL Fortran for AIX Version 5 and the Fortran 90 standard for certain aspects of the Fortran language.

### Defaults

The default suboptions for **-qxlf90** depend on the invocation command that you specify. For the **f95**, **xlf95**, **xlf95\_r**, or **xlf95\_r7** command, the default suboptions are **signedzero** and **autodealloc**. For all other invocation commands, the defaults are **nosignedzero** and **noautodealloc**.

### Arguments

#### signedzero | **nosignedzero**

Determines how the **SIGN(A,B)** function handles signed real 0.0. Prior to XL Fortran Version 6.1, **SIGN(A,B)** returned  $|A|$  when  $B=-0.0$ . This behavior conformed with the Fortran 90 standard. Now, if you specify the **-qxlf90=signedzero** compiler option, **SIGN(A,B)** returns  $-|A|$  when  $B=-0.0$ . This behavior conforms to the Fortran 95 standard and is consistent with the IEEE standard for binary floating-point arithmetic. Note that for the **REAL(16)** data type, XL Fortran never treats zero as negative zero.

This suboption also determines whether a minus sign is printed in the following cases:

- For a negative zero in formatted output. Again, note that for the **REAL(16)** data type, XL Fortran never treats zero as negative zero.
- For negative values that have an output form of zero (that is, where trailing non-zero digits are truncated from the output so that the resulting output looks like zero). Note that in this case, the **signedzero** suboption does affect the **REAL(16)** data type; non-zero negative values that have an output form of zero will be printed with a minus sign.

#### autodealloc | **noautodealloc**

Determines whether the compiler deallocates allocatable objects that are declared locally without either the **SAVE** or the **STATIC** attribute and have a status of currently allocated when the subprogram terminates. This behavior conforms with the Fortran 95 standard and did not exist in XL Fortran prior to Version 6.1. If you are certain that you are deallocating all local allocatable objects explicitly, you may wish to turn off this suboption to avoid possible performance degradation.

## Examples

Consider the following program:

```
PROGRAM TESTSIGN
REAL X, Y, Z
X=1.0
Y=-0.0
Z=SIGN(X,Y)
PRINT *,Z
END PROGRAM TESTSIGN
```

The output from this example depends on the invocation command and the **-qxf90** suboption that you specify. For example:

| Invocation Command/xlf90 Suboption | Output |
|------------------------------------|--------|
| xlf95                              | -1.0   |
| xlf95 -qxf90=signedzero            | -1.0   |
| xlf95 -qxf90=nosignedzero          | 1.0    |
| xlf90                              | 1.0    |
| xlf                                | 1.0    |

## Related information

See the **SIGN** information in the *Intrinsic Procedures* section and the *Arrays Concepts* section of the *XL Fortran Language Reference*.

## -qxlines option

### Syntax

-qxlines | -qnoxlines  
XLINES | NOXLINES

Specifies whether fixed source form lines with a X in column 1 are compiled or treated as comments. This option is similar to the recognition of the character 'd' in column 1 as a conditional compilation (debug) character. The **-D** option recognizes the character 'x' in column 1 as a conditional compilation character when this compiler option is enabled. The 'x' in column 1 is interpreted as a blank, and the line is handled as source code.

### Defaults

This option is set to **-qnoxlines** by default, and lines with the character 'x' in column 1 in fixed source form are treated as comment lines. While the **-qxlines** option is independent of **-D**, all rules for debug lines that apply to using 'd' as the conditional compilation character also apply to the conditional compilation character 'x'. The **-qxlines** compiler option is only applicable to fixed source form.

The conditional compilation characters 'x' and 'd' may be mixed both within a fixed source form program and within a continued source line. If a conditional compilation line is continued onto the next line, all the continuation lines must have 'x' or 'd' in column 1. If the initial line of a continued compilation statement is not a debugging line that begins with either 'x' or 'd' in column 1, subsequent continuation lines may be designated as debug lines as long as the statement is syntactically correct.

The OMP conditional compilation characters '!\$', 'C\$', and '\*\$' may be mixed with the conditional characters 'x' and 'd' both in fixed source form and within a continued source line. The rules for OMP conditional characters will still apply in this instance.

### Examples

An example of a base case of -qxlines:

```
C2345678901234567890
 program p
 i=3 ; j=4 ; k=5
X print *,i,j
X + ,k
 end program p

<output>: 3 4 5 (if -qxlines is on)
 no output (if -qxlines is off)
```

In this example, conditional compilation characters 'x' and 'd' are mixed, with 'x' on the initial line:

```
C2345678901234567890
 program p
 i=3 ; j=4 ; k=5
X print *,i,
D + j,
X + k
 end program p

<output>: 3 4 5 (if both -qxlines and -qdlines are on)
 3 5 (if only -qxlines is turned on)
```

Here, conditional compilation characters 'x' and 'd' are mixed, with 'd' on the initial line:

```
C2345678901234567890
 program p
 i=3 ; j=4 ; k=5
D print *,i,
X + j,
D + k
 end program p

<output>: 3 4 5 (if both -qxlines and -qdlines are on)
 3 5 (if only -qdlines is turned on)
```

In this example, the initial line is not a debug line, but the continuation line is interpreted as such, since it has an 'x' in column 1:

```
C2345678901234567890
 program p
 i=3 ; j=4 ; k=5
 print *,i
X + ,j
X + ,k
 end program p

<output>: 3 4 5 (if -qxlines is on)
 3 (if -qxlines is off)
```

### Related information

See “-D option” on page 91 and *Conditional Compilation* in the *Language Elements* section of the *XL Fortran Language Reference*.

## **-qxref option**

### **Syntax**

`-qxref[=full] | -qnoxref`  
`XREF[(FULL)] | NOXREF`

Determines whether to produce the cross-reference component of the attribute and cross-reference section of the listing.

If you specify only `-qxref`, only identifiers that are used are reported. If you specify `-qxref=full`, the listing contains information about all identifiers that appear in the program, whether they are used or not.

If `-qxref` is specified after `-qxref=full`, the full cross-reference listing is still produced.

You can use the cross-reference listing during debugging to locate problems such as using a variable before defining it or entering the wrong name for a variable.

### **Related information**

See “Options that control listings and messages” on page 62 and “Attribute and cross-reference section” on page 298.

## -qzerosize option

### Syntax

**-qzerosize** | **-qnozerosize**  
**ZEROSIZE** | **NOZEROSIZE**

Improves performance of FORTRAN 77 and some Fortran 90 and Fortran 95 programs by preventing checking for zero-sized character strings and arrays.

For Fortran 90 and Fortran 95 programs that might process such objects, use **-qzerosize**. For FORTRAN 77 programs, where zero-sized objects are not allowed, or for Fortran 90 and Fortran 95 programs that do not use them, compiling with **-qnozerosize** can improve the performance of some array or character-string operations.

### Defaults

The default setting depends on which command invokes the compiler: **-qzerosize** for the **xlF90**, **xlF90\_r**, **xlF90\_r7**, **xlF95**, **xlF95\_r**, **xlF95\_r7**, **f90**, and **f95** commands and **-qnozerosize** for the **xlF**, **xlF\_r**, **xlF\_r7**, and **f77/fort77** commands (for compatibility with FORTRAN 77).

### Rules

Run-time checking performed by the **-C** option takes slightly longer when **-qzerosize** is in effect.

## **-S option**

### **Syntax**

-S

Produces one or more `.s` files that show equivalent assembler source for each Fortran source file.

### **Rules**

When this option is specified, the compiler produces the assembler source files as output instead of an object or an executable file.

### **Restrictions**

The generated assembler files do not include all the data that is included in a `.o` file by `-qipa` or `-g`.

### **Examples**

```
xlf95 -O3 -qhot -S test.f # Produces test.s
```

### **Related information**

The “-o option” on page 102 can be used to specify a name for the resulting assembler source file.

For information about the assembler-language format, see the *Assembler Language Reference*.

## -t option

### Syntax

*-tcomponents*

Applies the prefix specified by the **-B** option to the designated components. *components* can be one or more of **a, F, p, c, h, I, b, z, l, or d** with no separators, corresponding to the assembler, the C preprocessor, an optimizing preprocessor, the compiler, the array language optimizer, the interprocedural analysis (IPA) tool/loop optimizer, the code generator, the binder, the linker, and the **-S** disassembler, respectively.

### Rules

If **-t** is not specified, any **-B** prefix is applied to all components.

| Component                | -t<br>Mnemonic | Standard Program Name |
|--------------------------|----------------|-----------------------|
| assembler                | a              | as                    |
| C preprocessor           | F              | cpp                   |
| VAST-2 preprocessor      | p              | fpp                   |
| KAP preprocessor         | p              | fppk                  |
| compiler front end       | c              | xlfcntry              |
| array language optimizer | h              | xlshot                |
| IPA/loop optimizer       | I              | ipa                   |
| code generator           | b              | xlfcg                 |
| binder                   | z              | bol                   |
| linker                   | l              | ld                    |
| disassembler             | d              | dis                   |

### Related information

See “-B option” on page 79 (which includes an example).

## **-U option**

### **Syntax**

`-U`  
`MIXED` | `NOMIXED`

Makes the compiler sensitive to the case of letters in names.

You can use this option when writing mixed-language programs, because Fortran names are all lowercase by default, while names in C and other languages may be mixed-case.

### **Rules**

If `-U` is specified, case is significant in names. For example, the names `Abc` and `ABC` refer to different objects.

The option changes the link names used to resolve calls between compilation units. It also affects the names of modules and thus the names of their `.mod` files.

### **Defaults**

By default, the compiler interprets all names as if they were in lowercase. For example, `Abc` and `ABC` are both interpreted as `abc` and so refer to the same object.

### **Restrictions**

The names of intrinsics must be all in lowercase when `-U` is in effect. Otherwise, the compiler may accept the names without errors, but the compiler considers them to be the names of external procedures, rather than intrinsics.

The XL Fortran Version 2 requirement that keywords be all lowercase no longer applies.

### **Related information**

This is the short form of `-qmixed`. See “`-qmixed` option” on page 190.

## **-u option**

### **Syntax**

-u  
UNDEF | NOUNDEF

Specifies that no implicit typing of variable names is permitted. It has the same effect as using the **IMPLICIT NONE** statement in each scope that allows implicit statements.

### **Defaults**

By default, implicit typing is allowed.

### **Related information**

See **IMPLICIT** in the *XL Fortran Language Reference*.

This is the short form of **-qundef**. See “-qundef option” on page 243.

## **-v option**

### **Syntax**

`-v`

Generates information on the progress of the compilation.

### **Rules**

As the compiler executes commands to perform different compilation steps, this option displays a simulation of the commands it calls and the system argument lists it passes.

For a particular compilation, examining the output that this option produces can help you determine:

- What files are involved
- What options are in effect for each step
- How far a compilation gets when it fails

### **Related information**

“`-#` option” on page 77 is similar to `-v`, but it does not actually execute any of the compilation steps.

## **-V option**

### **Syntax**

`-V`

This option is the same as `-v` except that you can cut and paste directly from the display to create a command.

## -W option

### Syntax

*-Wcomponent,options*

Passes the listed options to a component that is executed during compilation. *component* is one of **a**, **F**, **p**, **c**, **h**, **I**, **b**, **z**, **l**, or **d**, corresponding to the assembler, the C preprocessor, an optimizing preprocessor, the compiler, the array language optimizer, the interprocedural analysis (IPA) tool/loop optimizer, the code generator, the binder, the linker, and the **-S** disassembler, respectively.

| Component                | -W Mnemonic | Standard Program Name |
|--------------------------|-------------|-----------------------|
| assembler                | a           | as                    |
| C preprocessor           | F           | cpp                   |
| VAST-2 preprocessor      | p           | fpp                   |
| KAP preprocessor         | p           | fppk                  |
| compiler front end       | c           | xlfcntry              |
| array language optimizer | h           | xlshot                |
| IPA/loop optimizer       | I           | ipa                   |
| code generator           | b           | xlfcde                |
| binder                   | z           | bolt                  |
| linker                   | l           | ld                    |
| disassembler             | d           | dis                   |

In the string following the **-W** option, use a comma as the separator for each option, and do not include any spaces. For example:

```
-Wcomponent,option_1[,option_2,...,option_n]
```

### Background information

The primary purpose of this option is to construct sequences of compiler options to pass to one of the optimizing preprocessors. It can also be used to fine-tune the link-edit step by passing parameters to the **ld** command.

### Defaults

You do not need the **-W** option to pass most options to the linker: unrecognized command-line options, except **-q** options, are passed to it automatically. Only linker options with the same letters as compiler options, such as **-v** or **-S**, strictly require **-W** (or the **ldopts** stanza in the configuration file).

If you need to include a character that is special to the shell in the option string, precede the character with a backslash.

### Examples

See “Passing command-line options to the **ld** or **as** commands” on page 28.

You can use **\**, to embed a literal comma in the string supplied to the **-W** option.

In the following example, the **\**, embeds a literal comma in the **-WF** string and causes three arguments, rather than four, to be supplied to the C preprocessor.

```
$ xlf -qfree=f90 '-WF,-Dint1=1,-Dint2=2,-Dlist=3\,4' a.F
$ cat a.F
print *, int1
print *, int2
print *, list
end
```

The output from the program will be:

```
$./a.out
1
2
3 4
```

## **-w option**

### **Syntax**

`-w`

A synonym for the “-qflag option” on page 151. It sets **-qflag=e:e**, suppressing warning and informational messages and also messages generated by language-level checking.

## **-y option**

### **Syntax**

`-y{n | m | p | z}`  
`IEEE(Near | Minus | Plus | Zero)`

Specifies the rounding mode for the compiler to use when evaluating constant floating-point expressions at compile time. It is equivalent to the **-qieee** option.

### **Arguments**

**n**      Round to nearest.  
**m**      Round toward minus infinity.  
**p**      Round toward plus infinity.  
**z**      Round toward zero.

### **Related information**

See “-O option” on page 100 and “-qfloat option” on page 152.

**-y** is the short form of “-qieee option” on page 162.



---

## Chapter 6. Using XL Fortran in a 64-Bit Environment

The 64-bit environment addresses an 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 space and 64-bit integers. XL Fortran only supports applications that target the 64-bit Large Data Type (LDT) Application Binary Interface (ABI). Non-LDT environments, such as AIX version 4, are no longer supported.

To support larger executables that can be fit within a 64-bit address space, a separate, 64-bit object form is used to meet the requirements of 64-bit executables. The binder binds 64-bit objects to create 64-bit executables. Note that 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 attempts to explicitly load a 32-bit module
- A 32-bit executable that attempts to explicitly 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.

The XL Fortran compiler mainly provides 64-bit mode support through the compiler option **-q64** in conjunction with the compiler option **-qarch**. This combination determines the bit mode and instruction set for the target architecture. The **-q32** and **-q64** options take precedence over the setting of the **-qarch** option. The **-q64** option will win over a 32-bit mode only **-qarch** setting, and the compiler will upgrade the **-qarch** setting to something that will handle 64-bit mode. Conflicts between the **-q32** and **-q64** options are resolved by the "last option wins" rule. Setting **-qarch=com** will ensure future compatibility for applications in 32-bit mode. For 64-bit mode applications, use **-qarch=ppc64** to achieve the same effect for all present or future supported 64-bit mode systems. **-qarch** settings that target a specific architecture, like the **603**, **604**, **rs64a**, **rs64b**, **rs64c**, **pwr3**, **pwr4**, **pwr5**, **pwr5x**, **ppc970**, and **auto** settings will be more system-dependent.

---

### 64-Bit Large Data Type support

The 64-bit Large Data Type (LDT) Application Binary Interface (ABI), or 64-bit LDT ABI, increases scalability for 64-bit applications, while maintaining binary compatibility for existing 32-bit applications. To accomplish this, some system-derived types are increased from 32-bits to 64-bits. In addition, a new 64-bit magic number is being introduced in the XCOFF definition to identify object code files using the new 64-bit ABI.

The AIX 4.3 64-bit, non-LDT, ABI is no longer supported on AIX 5.1. Object code files with the old 64-bit magic number will not link, load, or execute. Pre-AIX 5.1 64-bit applications must be recompiled in order to execute them on AIX 5.1. Binary compatibility will be preserved for 32-bit objects generated on all earlier levels of AIX, regardless of LDT support.

---

## 64-Bit thread support

On AIX Version 5.1 with the POSIX 1003.1-1996 standard pthreads API, XL Fortran supports 64-bit thread programming. You can specify the **-q64** compiler option with the `xlf_r`, `xlf_r7`, `xlf90_r`, `xlf90_r7`, `xlf95_r`, and `xlf95_r7` commands. For example, you can specify the following command to compile and then link a program in 64-bit object mode:

```
xlf90_r -q64 -qsmp test.f
```

AIX Version 5.1 supports the POSIX 1003.1-1996 standard interface in both 32-bit and 64-bit object mode, but supports the Draft 7 interface in 32-bit object mode only. That is, the `libpthreads.a` library has a 32-bit and a 64-bit part, while the `libpthreads_compat.a` and `libxlfthrds_compat.a` libraries have 32-bit parts only.

---

## Compiler options for the 64-Bit environment

The **-q32**, **-q64**, and **-qwarn64** compiler options are primarily for developers who are targeting 64-bit platforms. They enable you to do the following:

- Develop applications for the 64-bit environment
- Help migrate source code from the 32-bit environment to a 64-bit environment

---

## Default bit mode

The AIX operating system provides support for the `OBJECT_MODE` environment variable to enable the user to obtain a 64-bit development environment. AIX tools use the setting of `OBJECT_MODE` to determine the type of object to be used or created. The `OBJECT_MODE` environment variable has three recognized settings:

**OBJECT\_MODE=32**

Works with 32-bit objects

**OBJECT\_MODE=64**

Works with 64-bit objects

**OBJECT\_MODE=32\_64**

Works with either 32-bit or 64-bit objects

The XL Fortran compiler determines the default bit mode through the setting of the `OBJECT_MODE` environment variable at the time of invocation. The following table shows the default bit mode and options that are set for each setting of the `OBJECT_MODE` environment variable:

*Table 20. Default bit mode determined by the setting of OBJECT\_MODE*

| OBJECT_MODE Setting | Default Bit Mode | Default Option Set |
|---------------------|------------------|--------------------|
| unset               | 32-bit           | -q32               |
| 32                  | 32-bit           | -q32               |
| 64                  | 64-bit           | -q64               |
| 32_64               | Not permitted    | n/a                |

Specification of the following options on the command line or in the configuration file overrides the default option set:

- **-q64**
- **-q32**

**Important note**

Using **OBJECT\_MODE** to determine the default bit mode can have serious implications if you are not aware of the setting of **OBJECT\_MODE** at the time of invocation. For example, you may not be aware that **OBJECT\_MODE** has been set to **64**, and you may unexpectedly obtain 64-bit object files.

We strongly urge you to be aware of the setting of **OBJECT\_MODE** at all times and to set **OBJECT\_MODE** yourself to ensure that the compiler is invoked for the correct bit mode. Instead, you can always use the **-q32** or **-q64** option to specify the bit mode.

**Module support**

64-bit support is provided in the Fortran module files that are shipped with XL Fortran.



---

## Chapter 7. Problem determination and debugging

This section describes some methods you can use for locating and fixing problems in compiling or executing your programs.

**Related information:** You might encounter a number of potential problems when moving from previous versions of XL Fortran to XL Fortran Version 10. “Avoiding or fixing upgrade problems” on page 16 summarizes these potential problems.

---

### Understanding XL Fortran error messages

Most information about potential or actual problems comes through messages from the compiler or application program. These messages are written to the standard error output stream.

#### Error severity

Compilation errors can have the following severity levels, which are displayed as part of some error messages:

- U** An unrecoverable error. Compilation failed because of an internal compiler error.
- S** A severe error. Compilation failed due to one of the following:
  - An unrecoverable program error has been detected. Processing of the source file stops, and XL Fortran does not produce an object file. You can usually correct this error by fixing any program errors that were reported during compilation.
  - Conditions exist that the compiler could not correct. An object file is produced; however, you should not attempt to run the program.
  - An internal compiler table has overflowed. Processing of the program stops, and XL Fortran does not produce an object file.
  - An include file does not exist. Processing of the program stops, and XL Fortran does not produce an object file.
- E** An error that the compiler can correct. The program should run correctly.
- W** Warning message. It does not signify an error but may indicate some unexpected condition.
- L** Warning message that was generated by one of the compiler options that check for conformance to various language levels. It may indicate a language feature that you should avoid if you are concerned about portability.
- I** Informational message. It does not indicate any error, just something that you should be aware of to avoid unexpected behavior or to improve performance.

**Notes:**

1. The message levels **S** and **U** indicate a compilation failure.
2. The message levels **I**, **L**, **W**, and **E** indicate that compilation was successful.

By default, the compiler stops without producing output files if it encounters a severe error (severity **S**). You can make the compiler stop for less severe errors by specifying a different severity with the **-qhalt=e** option. For example, with **-qhalt=e**,



|                        |                                                                             |
|------------------------|-----------------------------------------------------------------------------|
| <b>01</b>              | Indicates an XL Fortran common message                                      |
| <b>11-20</b>           | Indicates a Fortran-specific message                                        |
| <b>24</b>              | Indicates a VAST preprocessor message                                       |
| <b>25</b>              | Indicates a run-time message from an XL Fortran application program         |
| <b>26</b>              | Indicates a KAP preprocessor message                                        |
| <b>85</b>              | Indicates a loop-transformation message                                     |
| <b>86</b>              | Indicates an interprocedural analysis (IPA) message                         |
| <i>nmn</i>             | Is the message number                                                       |
| <i>severity_letter</i> | Indicates how serious the problem is, as described in the preceding section |
| <i>'message text'</i>  | Is the text describing the error                                            |

## Limiting the number of compile-time messages

If the compiler issues many low-severity (**I** or **W**) messages concerning problems you are aware of or do not care about, use the **-qflag** option or its short form **-w** to limit messages to high-severity ones:

```
E, S, and U messages go in listing; U messages are displayed on screen.
xlf95 -qflag=e:u program.f
```

```
E, S, and U messages go in listing and are displayed on screen.
```

```
xlf95 -w program.f
```

## Selecting the language for messages

By default, XL Fortran comes with messages in U.S. English only. You can also order translated message catalogs:

- Compiler messages in Japanese
- Run-time messages in Japanese

If compile-time messages are appearing in U.S. English when they should be in another language, verify that the correct message catalogs are installed and that the **LANG**, **LC\_MESSAGES**, and/or **LC\_ALL** environment variables are set accordingly.

If a run-time message appears in the wrong language, also ensure that your program calls the **setlocale** routine.

**Related information:** See “Environment variables for national language support” on page 9 and “Selecting the language for run-time messages” on page 40.

To determine which XL Fortran message catalogs are installed, use the following commands to list them:

```
lsipp -f 'xlfcmp.msg.*' # compile-time messages
lsipp -f 'xlfrte.msg.*' # run-time messages
```

The file names of the message catalogs are the same for all supported international languages, but they are placed in different directories.

**Note:** When you run an XL Fortran program on a system without the XL Fortran message catalogs, run-time error messages (mostly for I/O problems) are not displayed correctly; the program prints the message number but not the associated text. To prevent this problem, copy the XL Fortran message catalogs from `/usr/lpp/xlf/bin/default_msg` to a directory that is part of the `NLSPATH` environment-variable setting on the execution system.

---

## Fixing installation or system environment problems

If individual users or all users on a particular machine have difficulty running the compiler, there may be a problem in the system environment. Here are some common problems and solutions:

---

xlF90: not found  
xlF90\_r: not found  
xlF90\_r7: not found  
xlF95: not found  
xlF95\_r: not found  
xlF95\_r7: not found  
xlF: not found  
xlF\_r: not found  
xlF\_r7: not found  
f77: not found  
fort77: not found  
f90: not found  
f95: not found

**Symptom:** The shell cannot locate the command to execute the compiler.

**Solution:** Make sure that your `PATH` environment variable includes the directory `/usr/bin`. If the compiler is properly installed, the commands you need to execute it are in this directory.

---

**Could not load program** *program*

**Error was:** not enough space

**Killed**

**Symptom:** The system cannot execute the compiler or an application program at all.

**Solution:** Set the storage limits for stack and data to “unlimited” for users who experience this problem. For example, as superuser you can set both your hard and soft limits with these `ksh` commands:

```
ulimit -s unlimited
ulimit -d unlimited
```

Because non-superusers are not completely free to give themselves unlimited limits, if you are a superuser you may find it more convenient to edit the file `/etc/security/limits` to give all users unlimited stack and data segments (by entering -1 for these fields).

If the storage problem is in an XLF-compiled program, using the `-qsave` or `-qsmallstack` option might prevent the program from exceeding the stack limit.

**Explanation:** The compiler allocates large internal data areas that may exceed the storage limits for a user. XLF-compiled programs place more data on the stack by default than in previous versions, also possibly

---

exceeding the storage limit. Because it is difficult to determine precise values for the necessary limits, we recommend making them unlimited.

---

**Could not load program** *program*

**Could not load library** *library\_name.a*

*[object\_name]*

**Error was:** no such file or directory

**Solution:** Make sure the XL Fortran libraries are installed in `/usr/lib`, or set the `LIBPATH` environment variable to include the directory where `libxlF90.a` is installed if it is in a different directory. See “`LIBPATH:Setting library search paths`” on page 10 for details of this environment variable.

---

**Symptom:** Messages from the compiler or an XL Fortran application program are displayed in the wrong language.

**Solution:** Set the appropriate national language environment. You can set the national language for each user with the command `smit chlang`. Alternatively, each user can set one or more of the environment variables `LANG`, `NLSPATH`, `LC_MESSAGES`, `LC_TIME`, and `LC_ALL`. If you are not familiar with the purposes of these variables, “`Environment variables for national language support`” on page 9 provides details.

---

**Symptom:** A compilation fails with an I/O error.

**Solution:** Increase the size of the `/tmp` filesystem, or set the environment variable `TMPDIR` to the path of a filesystem that has more free space.

**Explanation:** The object file may have grown too large for the filesystem that holds it. The cause could be a very large compilation unit or initialization of all or part of a large array in a declaration.

---

**Symptom:** There are too many individual makefiles and compilation scripts to easily maintain or track.

**Solution:** Add stanzas to the configuration file, and create links to the compiler by using the names of these stanzas. By running the compiler with different command names, you can provide consistent groups of compiler options and other configuration settings to many users.

---

## Fixing compile-time problems

The following sections discuss common problems you might encounter while compiling and how to avoid them.

## Duplicating extensions from other systems

Some ported programs may cause compilation problems because they rely on extensions that exist on other systems. XL Fortran supports many extensions like these, but some require compiler options to turn them on. See “Options for compatibility” on page 64 for a list of these options and *Porting programs to XL Fortran* in the *XL Fortran Optimization and Programming Guide* for a general discussion of porting.

## Isolating problems with individual compilation units

If you find that a particular compilation unit requires specific option settings to compile properly, you may find it more convenient to apply the settings in the source file through an **@PROCESS** directive. Depending on the arrangement of your files, this approach may be simpler than recompiling different files with different command-line options.

## Compiling with thread-safe commands

Thread-safe invocation commands, like **xl\_f\_r** or **xl\_f90\_r**, for example, use different search paths and call different modules than the non thread-safe invocations. Your programs should account for the different usages. Programs that compile and run successfully for one environment may produce unexpected results when compiled and run for a different use. The configuration file, **xl\_f.cfg**, shows the paths, libraries, and so on for each invocation command. (See “Customizing the configuration file” on page 11 for a sample configuration file and an explanation of its contents.)

## Running out of machine resources

If the operating system runs low on resources (page space or disk space) while one of the compiler components is running, you should receive one of the following messages:

```
1501-229 Compilation ended because of lack of space.
1501-224 fatal error in /usr/lpp/xlf/bin/xlfentry: signal 9 received.
1517-011 Compilation ended. No more system resources available.
Killed.
1501-053 (S) Too much initialized data.
1501-511. Compilation failed for file [filename].
```

You may need to increase the system page space and recompile your program. See *AIX General Concepts and Procedures* for more information about page space.

If your program produces a large object file, for example, by initializing all or part of a large array, you may need to do one of the following:

- Increase the size of the filesystem that holds the **/tmp** directory.
- Set the **TMPDIR** environment variable to a filesystem with a lot of free space.
- For very large arrays, initialize the array at run time rather than statically (at compile time).

---

## Fixing link-time problems

After the XL Fortran compiler processes the source files, the linker links the resulting object files together. Any messages issued at this stage come from the **ld** or **bind** commands. A frequently encountered error and its solution are listed here for your convenience:

---

**0706-317 ERROR: Undefined or unresolved symbols detected:**

**Symptom:** A program cannot be linked because of unresolved references.

**Explanation:** Either needed object files or libraries are not being used during linking, there is an error in the specification of one or more external names, or there is an error in the specification of one or more procedure interfaces.

**Solution:** You may need to do one or more of the following actions:

- Compile again with the **-bloadmap** option to create a file that contains information about undefined symbols.
- Make sure that if you use the **-U** option, all intrinsic names are in lowercase.
- Use the linker **-brename** option on the compiler command line to change the names of some symbols at link time.

---

## Fixing run-time problems

XL Fortran issues error messages during the running of a program in either of the following cases:

- XL Fortran detects an input/output error. “Setting run-time options” on page 40 explains how to control these kinds of messages.
- XL Fortran detects an exception error, and the default exception handler is installed (through the **-qsigtrap** option or a call to **SIGNAL**). To get a more descriptive message than Core dumped, you may need to run the program from within **dbx**.

The causes for run-time exceptions are listed in “XL Fortran run-time exceptions” on page 49.

You can investigate errors that occur during the execution of a program by using a symbolic debugger, such as **dbx**.

## Duplicating extensions from other systems

Some ported programs may not run correctly if they rely on extensions that are found on other systems. XL Fortran supports many such extensions, but you need to turn on compiler options to use some of them. See “Options for compatibility” on page 64 for a list of these options and *Porting programs to XL Fortran* in the *XL Fortran Optimization and Programming Guide* for a general discussion of porting.

## Mismatched sizes or types for arguments

To detect arguments of different sizes or types, which might produce incorrect execution and results, you can compile with the **-qextchk** option. This option warns you of any problems at link time.

To do the type-checking during the early stages of compilation, specify interface blocks for the procedures that are called within a program.

## Working around problems when optimizing

If you find that a program produces incorrect results when it is optimized and if you can isolate the problem to a particular variable, you might be able to work around the problem temporarily by declaring the variable as **VOLATILE**. This prevents some optimizations that affect the variable. (See **VOLATILE** in the *XL Fortran Language Reference*.) Because this is only a temporary solution, you should continue debugging your code until you resolve your problem, and then remove the **VOLATILE** keyword. If you are confident that the source code and program design are correct and you continue to have problems, contact your support organization to help resolve the problem.

## Input/Output errors

If the error detected is an input/output error and you have specified **IOSTAT** on the input/output statement in error, the **IOSTAT** variable is assigned a value according to *Conditions and IOSTAT Values* in the *XL Fortran Language Reference*.

If you have installed the XL Fortran run-time message catalog on the system on which the program is executing, a message number and message text are issued to the terminal (standard error) for certain I/O errors. If you have specified **IOMSG** on the input/output statement, the **IOMSG** variable is assigned the error message text if an error is detected, or the content of **IOMSG** variable is not changed. If this catalog is not installed on the system, only the message number appears. Some of the settings in “Setting run-time options” on page 40 allow you to turn some of these error messages on and off.

If a program fails while writing a large data file, you may need to increase the maximum file size limit for your user ID. You can do this through a shell command, such as **ulimit** in **ksh**, or through the **smit** command.

## Tracebacks and core dumps

If a run-time exception occurs and an appropriate exception handler is installed, a message and a traceback listing are displayed. Depending on the handler, a core file might be produced as well. You can then use a debugger to examine the location of the exception.

To produce a traceback listing without ending the program, call the **xl\_\_trbk** procedure:

```
IF (X .GT. Y) THEN ! X > Y indicates that something is wrong.
 PRINT *, 'Error - X should not be greater than Y'
 CALL XL__TRBK ! Generate a traceback listing.
END IF
```

See *Installing an exception handler* in the *XL Fortran Optimization and Programming Guide* for instructions about exception handlers and “XL Fortran run-time exceptions” on page 49 for information about the causes of run-time exceptions.

---

## Debugging a Fortran 90 or Fortran 95 program

You can use **dbx** and other symbolic debuggers to debug your programs. For instructions on using your chosen debugger, consult the online help within the debugger or its documentation.

Always specify the **-g** option when compiling programs for debugging.

### Related information:

- “Options for error checking and debugging” on page 59
- “A Sample dbx session for an XL Fortran program”

---

## A Sample dbx session for an XL Fortran program

You can debug XL Fortran programs with any **dbx**-compatible symbolic debugger. For background information on **dbx**, see the *AIX General Concepts and Procedures* document. For information on **dbx** subcommands, see the *AIX Commands Reference*.

The following example represents a typical XL Fortran problem that you may be able to resolve through **dbx**. Although it demonstrates only a small subset of **dbx** features and uses memory-allocation techniques made obsolete by Fortran 90/Fortran 95 allocatable arrays, it can serve as an introduction if you have not used this debugger before.

## Problem with dynamic memory allocation

The following program tries to allocate an array at run time by using the AIX system subroutine **malloc**. When you use the following command to compile the program and then run the program, the program produces a core dump:

```
xlf95 -qddim testprog.f -o testprog
```

At this point, you may be wondering whether the C **malloc** routine is working correctly or whether this is the right way to allocate an array in a main program when the dimensions are not known until run time.

```
program main

pointer(p, array(nvar,nrec))
real*8 array

nvar = 2
nrec = 3
p = malloc(nvar*nrec*8)

call test_sub(array, nvar, nrec)

end

subroutine test_sub(array, nvar, nrec)

dimension array(nvar, nrec)

array(1,1) = 1.
array(2,1) = 2.
array(1,2) = 3.
array(2,2) = 4.
array(1,3) = 5.
array(2,3) = 6.

write(*, 100) array(1,1), array(2,1), array(1,2),
1 array(2,2), array(1,3), array(2,3)
100 format(//t2,f4.1/t2,f4.1/t2,f4.1/t2,f4.1/t2,f4.1/
1 t2,f4.1/t2,f4.1)

return
end
```

You might go through the debugging process as follows:

1. Compile the program with the **-g** option, to allow debugging under **dbx**:

```
-> xlf95 -qddim -g testprog.f -o testprog
** main === End of Compilation 1 ===
** test_sub === End of Compilation 2 ===
1501-510 Compilation successful for file testprog.f.
```

2. Run the program to verify the problem and create a core dump:

```
-> testprog
Segmentation fault(coredump)
->
```

3. Find out where in the program the core dump occurs:

```
-> dbx testprog core
dbx version 3.1 for AIX.
Type 'help' for help.
reading symbolic information ...
[using memory image in core]

segmentation violation in test_sub at line 21 in file "testprog.f"
 21 array(1,1) = 1.
(dbx)
```

4. Use the **where** command to get a traceback of the calls that led to that point in the program:

```
(dbx) where
test_sub(array = (...), nvar = warning: Unable to access address 0x200aee94
from core
-1, nrec = warning: Unable to access address 0x200aee98 from core
-1), line 21 in "testprog.f"
main(), line 12 in "testprog.f"
(dbx)
```

**main** calls **test\_sub** at line 12. The warning indicates that a problem occurs while evaluating the arguments for this call.

5. Look at the value of the first argument of the array:

```
(dbx) print array(1,1)
reference through nil pointer
(dbx)
```

This suggests that **array** does not have a value assigned. To verify that possibility, try to look at the address of an element in the array:

```
(dbx) p &array(1,1)
(nil)
(dbx)
```

It seems that XL Fortran has not allocated the space for the array. To verify that it has not, print the value of the pointer that points to the array:

```
(dbx) print p
warning: Unable to access address 0x200aee90 from core
0xffffffff
```

6. To find out what happens to **p** during execution, restart the program and trace the use of **p**:

```
(dbx) stop in main
[1] stop in main
(dbx) run
[1] stopped in main at line 7 in file "testprog.f"
 7 nvar = 2
(dbx) trace p
[3] trace p
(dbx) cont
initially (at line 8 in "testprog.f"): p = nil

segmentation violation in test_sub at line 21 in file "testprog.f"
 21 array(1,1) = 1.
(dbx) p p
nil
(dbx)
```

Because **p** is never set to a valid value, something must be wrong with the line that allocates space for the array:

```
9 p = malloc(nvar*nrec*8)
```

7. The next step is to research why the call to **malloc** does not work. Because **malloc** is a C function, you should read *Interlanguage calls* in the *XL Fortran Optimization and Programming Guide* for background knowledge and specific guidelines.

When you read that section, you find that calls to C functions require arguments to be passed by value, rather than by reference. To fix the problem in this sample program, replace the line:

```
p = malloc(nvar*nrec*8)
```

with the line:

```
p = malloc(%val(nvar*nrec*8))
```

8. Compiling and running the fixed program (**solution.f**) again produces the correct result:

```
-> xlf95 -qddim -g solution.f -o solution
** main === End of Compilation 1 ===
** test_sub === End of Compilation 2 ===
1501-510 Compilation successful for file solution.f.
-> solution
```

```
1.0
2.0
3.0
4.0
5.0
6.0
```

9. It might be informative to trace the execution of the corrected program:

```
-> dbx solution
dbx version 3.1 for AIX.
Type 'help' for help.
Core file program (testprog) does not match current program (core ignored)
reading symbolic information ...
(dbx) trace p
[1] trace p
(dbx) run
initially (at line 7 in "solution.f"): p = nil
after line 9 in "solution.f": p = 0x200af100
```

```
1.0
2.0
3.0
4.0
5.0
6.0
```

```
execution completed
(dbx)
```

To check whether the values of **p** and **array** are appropriate, turn off the trace:

```
(dbx) status
[1] trace p
(dbx) delete all
(dbx) status
(dbx)
```

Then set new break points and run through the program again. Notice that the address of **array(1,1)** is the same as the contents of **p(0x200af100)**, as expected:

```
(dbx) stop at 9
[11] stop at "solution.f":9
(dbx) run
[11] stopped in main at line 9 in file "solution.f"
 9 p = malloc(%val(nvar*nrec*8))
(dbx) p p
nil
(dbx) next
stopped in main at line 12 in file "solution.f"
 12 call test_sub(array, nvar, nrec)
(dbx) p p
0x200af100 <-----
(dbx)
(dbx) step /* Notice we use step to step into subroutine test_sub. */
stopped in test_sub at line 21 in file "solution.f"
 21 array(1,1) = 1.
(dbx) p &array(1,1)
0x200af100 <-----
(dbx) next
stopped in test_sub at line 22 in file "solution.f"
 22 array(2,1) = 2.
(dbx) p array(1,1)
1.0
(dbx)
```

---

## Using debug memory routines for XL Fortran

The XL Fortran compiler contains two libraries that are geared to various memory-allocation facilities. These libraries include:

- libhmd.a**      A library that provides debug versions of memory-management routines.
- libhm.a**      A non-debug library that provides replacement routines for **malloc**, **free**, and so on. These routines are faster than the usual AIX versions. In addition, this library contains a few new library routines to provide additional facilities for memory management and production-level heap error checking.

The library of most interest to Fortran users is **libhmd.a**. See “The libhmd.a library” on page 289 for additional details. If you are installing an application built with these libraries in an environment that does not have XL Fortran installed, you may need to include the library **libhu.a** as well. This is because some routines in **libhmd.a** and **libhm.a** are dependent on routines in **libhu.a**.

### The libhm.a library

**libhm.a** provides fast replacement routines for the following **libc.a** procedures: **malloc**, **calloc**, **realloc**, **free**, **strdup**, **mallopt**, and **mallinfo**. The interfaces to these routines are exactly the same as the interfaces to the standard system routines, so a user need only link in **libhm.a** before the system libraries to make use of them.

In addition, the following library routines that are provided in **libhm.a** are available to Fortran users: **\_heapchk** and **\_heapset**. These routines provide production-level services that assist in maintaining consistent and correct heap storage.

Note that you cannot use the **-qextname** compiler option with programs that use these facilities. In other words, since these library routines are "system-like" routines that are not Fortran-specific, we do not provide "\_" versions of the routines in our library.

The following table describes the additional routines that you can use from **libhm.a**:

| C Function prototype                 | Fortran usage example                                                                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------------------------|-------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int _heapchk(void);                  | integer(4) _heapchk,<br>retc<br>retc = _heapchk()                                         | Does consistency checking for all allocated and freed objects on the heap.<br><br>Return values:<br><b>0</b> The heap is consistent.<br><b>1</b> Reserved.<br><b>2</b> Heap errors have occurred.                                                                                                                                                                                                                                                                                                  |
| int _heapset<br>(unsigned int fill); | integer(4) _heapset,<br>retc<br>integer(4) fill /1/<br><br>retc =<br>_heapset(%val(fill)) | <b>_heapset</b> checks the heap for consistency (similar to <b>_heapchk</b> ). It then sets each <i>byte</i> of any non-reserved freed storage to the value of <i>fill</i> . The value of <i>fill</i> must be an integer in the range of 0-255.<br><br>Using <b>_heapset</b> can help a user locate problems where a program continues to use a freed pointer to an object.<br><br>Return values:<br><b>0</b> The heap is consistent.<br><b>1</b> Reserved.<br><b>2</b> Heap errors have occurred. |

### Examples:

#### Example 1: Using \_heapchk to test for heap errors

```

program tstheapchk
pointer (p,pbased),(q,qbased)
integer pbased,qbased

integer(4) _heapchk,retcode

p = malloc(%val(4))
pbased = 10

! Decrement the pointer and store into
! memory we do not own.
q = p-4;
qbased = 10

retcode = _heapchk()
if (retcode .ne. 0) call abort()

```



- Memory leak reporting that indicates where the allocation of non-freed storage occurred and that displays partial contents of the area that is not freed.
- Memory error detection that includes:
  - Freeing the same location multiple times
  - Overwriting the end of an allocated object (note that the **-qcheck** compiler option available with XL Fortran already provides much of this functionality)
  - Reading data from or writing data to a freed object
  - Freeing an invalid pointer

You obtain access to this functionality when you link in the **libhmd.a** library prior to the system libraries. References to **malloc**, **realloc**, and **free** can be explicit, or you can obtain heap debugging for memory allocated and deallocated via the **ALLOCATE** and **DEALLOCATE** statements. To obtain source line number information in the output that the debug library produces, you should compile with the **-g** compiler option.

Note that you cannot specify the **-qextname** compiler option with programs that use these facilities.

The following shows the external interface and description of the procedures that are provided. Note that the external interfaces and functionality of **malloc**, **free**, **calloc**, **realloc**, and **strdup** are not shown in this table, since they have not changed.

| C Function prototype                        | Fortran usage example                                                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------------------------------------|-------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>void _dump_allocated (int size);</pre> | <pre>integer(4) :: size=4 call _dump_allocated &amp; (%val(size))</pre> | <p>This routine prints information to stderr about each memory block that is currently allocated or was allocated using the debug memory management routines. <i>size</i> indicates how many bytes of each memory block are to be printed, as follows:</p> <p><b>Negative size</b> All bytes are displayed.</p> <p><b>0 size</b> No bytes are displayed.</p> <p><b>Positive size</b> Specified number of bytes are displayed.</p> |

| C Function prototype                              | Fortran usage example                                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------------------|-------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>void _dump_allocated_delta (int size);</pre> | <pre>integer(4) :: size=4 call _dump_allocated_delta &amp; (%val(size))</pre> | <p>This routine prints information to stderr about each memory block that is currently allocated or was allocated using the debug memory management routines since the last call to <code>_dump_allocated</code> or <code>_dump_allocated_delta</code>. <i>size</i> indicates how many bytes of each memory block are to be printed, as follows:</p> <p><b>Negative size</b> All bytes are displayed.</p> <p><b>0 size</b> No bytes are displayed.</p> <p><b>Positive size</b> Specified number of bytes are displayed.</p>                                                                                                                                                                                                                                                                                                                                                                    |
| <pre>void heap_check(void);</pre>                 | <pre>call _heap_check()</pre>                                                 | <p>This routine does consistency checking for memory blocks that have been allocated using the debug memory-management routines. It checks that your program has not overwritten freed storage or memory outside the bounds of allocated blocks.</p> <p>All of the debug memory-allocation routines (debug version of malloc, and so on) invoke <code>heap_check</code> automatically. It may also be invoked explicitly in areas of code where a user believes there may be memory problems.</p> <p>Calling <code>heap_check</code> frequently can increase the memory requirements and affect the performance of a program. The <b>HD_SKIP</b> environment variable may be used to control how often the debug memory procedures check the heap.</p> <p>Note that the errors are detected when the <code>heap_check</code> routine is invoked, not at the point of error in the program.</p> |

## Environment variables

The debug libraries support the following environment variables:

**HD\_SKIP=increment** [*start*] Control how often `heap_check` is invoked from the debug versions of the memory-management routines. *increment* indicates how often you want the debug functions to check the heap. *start* specifies that the skipping of heap checks should begin after the debug memory routines have been

called a certain number of times. The default values for *increment* and *start* are 1 and 0, respectively.

## HD\_FILL

When this environment variable is exported, the debug versions of *malloc* and *realloc* set the memory allocated to a byte pattern of 0xAA.

## HD\_STACK=*n*

*n* specifies how many procedures should appear in the call chain that the debug memory routines produce. The default is 10, or it is the number of routines in the call chain, if fewer than 10.

For example:

```
export HD_SKIP=10
! Every 10th debug memory function calls heap_check.

export HD_SKIP=100,10
! After 100 calls to debug memory functions, every 10th call
! will result in a call to heap_check.
```

## Examples:

### Example 1: Memory leak detection

```
pointer (p,a),(p2,b),(p3,c) ! 1
character a(4) ! 2
integer b,c ! 3
! 4
p = malloc(%val(4)) ! 5
a(1) = 'a' ! 6
a(2) = 'b' ! 7
a(3) = 'c' ! 8
a(4) = 'd' ! 9
! 10
p2 = malloc(%val(4)) ! 11
b = 1 ! 12
! 13
call _dump_allocated(%val(4)) ! 14
! 15
p3 = malloc(%val(4)) ! 16
c = 2 ! 17
! 18
call _dump_allocated_delta(%val(4)) ! 19
end ! 20
```

Output:

```
1546-515 -----
1546-516 START OF DUMP OF ALLOCATED MEMORY BLOCKS
1546-515 -----
1546-518 Address: 0x20000DE0 Size: 0x00000004 (4)
 _int_debug_umalloc + 32C
 _debug_umalloc + 44
 _dbg_umalloc + 18
 _umalloc_init + 30
 malloc + 24
 _main + 24 [x.f:5]
 1000022C
1546-520 Memory contents: 61626364 [abcd]
1546-515 -----
1546-518 Address: 0x2000DE10 Size: 0x00000004 (4)
 _int_debug_umalloc + 32C
 _debug_umalloc + 44
 _dbg_umalloc + 18
```

```

 malloc + 24
 _main + 64 [x.f:11]
 1000022C
1546-520 Memory contents: 00000001 [....]
1546-515 -----
1546-517 END OF DUMP OF ALLOCATED MEMORY BLOCKS
1546-515 -----
1546-515 -----
1546-516 START OF DELTA DUMP OF ALLOCATED MEMORY BLOCKS
1546-515 -----
1546-518 Address: 0x2000DE30 Size: 0x00000004 (4)
 _int_debug_umalloc + 32C
 _debug_umalloc + 44
 _dbg_umalloc + 18
 malloc + 24
 _main + 8C [x.f:16]
 1000022C
1546-520 Memory contents: 00000002 [....]
1546-515 -----
1546-517 END OF DELTA DUMP OF ALLOCATED MEMORY BLOCKS
1546-515 -----

```

**Example 2: Invalid write**

```

pointer (p,a) ! 1
integer a ! 2
 ! 3
p = malloc(%val(4)) ! 4
a = 1 ! 5
p = p + 4 ! 6
a = 2 ! 7
 ! 8
call _heap_check() ! 9
 ! 10
end ! 11

```

Output:

```

1546-503 End of allocated object 0x20000BD0 was overwritten at 0x20000BD4.
1546-514 The first eight bytes of the object (in hex) are: 0000000100000002.
 _int_debug_umalloc + 32C
 _debug_umalloc + 44
 _dbg_umalloc + 18
 _umalloc_init + 30
 malloc + 24
 _main + 24 [x.f:4]
 1000022C
1546-522 Traceback:
 0xD09D1C94 = _uheap_check_init + 0x24
 0xD09D18C0 = heap_check + 0x28
 0x100002C8 = _main + 0x5C
IOT/Abort trap(coredump)

```



---

## Chapter 8. Understanding XL Fortran compiler listings

Diagnostic information is placed in the output listing produced by the **-qlist**, **-qsource**, **-qxref**, **-qattr**, **-qreport**, and **-qlistopt** compiler options. The **-S** option generates an assembler listing in a separate file.

To locate the cause of a problem with the help of a listing, you can refer to the following:

- The source section (to see any compilation errors in the context of the source program)
- The attribute and cross-reference section (to find data objects that are misnamed or used without being declared or to find mismatched parameters)
- The transformation and object sections (to see if the generated code is similar to what you expect)

A heading identifies each major section of the listing. A string of greater than symbols precedes the section heading so that you can easily locate its beginning:

```
>>>> section name
```

You can select which sections appear in the listing by specifying compiler options.

**Related information:** See “Options that control listings and messages” on page 62.

---

### Header section

The listing file has a header section that contains the following items:

- A compiler identifier that consists of the following:
  - Compiler name
  - Version number
  - Release number
  - Modification number
  - Fix number
- Source file name
- Date of compilation
- Time of compilation

The header section is always present in a listing; it is the first line and appears only once. The following sections are repeated for each compilation unit when more than one compilation unit is present.

---

### Options section

The options section is always present in a listing. There is a separate section for each compilation unit. It indicates the specified options that are in effect for the compilation unit. This information is useful when you have conflicting options. If you specify the **-qlistopt** compiler option, this section lists the settings for all options.

---

## Source section

The source section contains the input source lines with a line number and, optionally, a file number. The file number indicates the source file (or include file) from which the source line originated. All main file source lines (those that are not from an include file) do not have the file number printed. Each include file has a file number associated with it, and source lines from include files have that file number printed. The file number appears on the left, the line number appears to its right, and the text of the source line is to the right of the line number. XL Fortran numbers lines relative to each file. The source lines and the numbers that are associated with them appear only if the **-qsource** compiler option is in effect. You can selectively print parts of the source by using the **@PROCESS** directives **SOURCE** and **NOSOURCE** throughout the program.

## Error messages

If the **-qsource** option is in effect, the error messages are interspersed with the source listing. The error messages that are generated during the compilation process contain the following:

- The source line
- A line of indicators that point to the columns that are in error
- The error message, which consists of the following:
  - The 4-digit component number
  - The number of the error message
  - The severity level of the message
  - The text that describes the error

For example:

```
 2 | equivalence (i,j,i)
 | a.
a - 1514-117: (E) Same name appears more than once in an
equivalence group.
```

If the **-qnosource** option is in effect, the error messages are all that appear in the source section, and an error message contains:

- The file name in quotation marks
- The line number and column position of the error
- The error message, which consists of the following:
  - The 4-digit component number
  - The number of the error message
  - The severity level of the message
  - The text that describes the error

For example:

```
"doc.f", line 6.11: 1513-039 (S) Number of arguments is not
permitted for INTRINSIC function abs.
```

---

## Transformation report section

If the **-qreport** option is in effect, a transformation report listing shows how XL Fortran optimized the program. This section displays pseudo-Fortran code that corresponds to the original source code, so that you can see parallelization and loop transformations that the **-qhot** and/or **-qsmp** options have generated.

### Sample Report

The following report was created for the program **t.f** using the `xlf -qhot -qreport t.f`

command.

#### Program t.f:

```
integer a(100, 100)
integer i,j

do i = 1 , 100
 do j = 1, 100
 a(i,j) = j
 end do
end do
end
```

#### Transformation Report:

>>>> SOURCE SECTION <<<<<

```
** _main === End of Compilation 1 ===
```

>>>> LOOP TRANSFORMATION SECTION <<<<<

```

 PROGRAM _main ()
4| IF (.FALSE.) GOTO lab_9
 @LoopIV0 = 0
 Id=1 DO @LoopIV0 = @LoopIV0, 99
5| IF (.FALSE.) GOTO lab_11
 @LoopIV1 = 0
 Id=2 DO @LoopIV1 = @LoopIV1, 99
 ! DIR_INDEPENDENT loopId = 0
6| a((@LoopIV1 + 1),(@LoopIV0 + 1)) = (@LoopIV0 + 1)
7| ENDDO
 lab_11
8| ENDDO
 lab_9
9| END PROGRAM _main
```

| Source File | Source Line | Loop Id | Action / Information                     |
|-------------|-------------|---------|------------------------------------------|
| -----       | -----       | -----   | -----                                    |
| 0           | 4           | 1       | Loop interchanging applied to loop nest. |

>>>> FILE TABLE SECTION <<<<<

---

## Attribute and cross-reference section

This section provides information about the entities that are used in the compilation unit. It is present if the `-qxref` or `-qattr` compiler option is in effect. Depending on the options in effect, this section contains all or part of the following information about the entities that are used in the compilation unit:

- Names of the entities
- Attributes of the entities (if `-qattr` is in effect). Attribute information may include any or all of the following details:
  - The type
  - The class of the name
  - The relative address of the name
  - Alignment
  - Dimensions
  - For an array, whether it is allocatable
  - Whether it is a pointer, target, or integer pointer
  - Whether it is a parameter
  - Whether it is volatile
  - For a dummy argument, its intent, whether it is value, and whether it is optional
  - Private, public, protected, module
- Coordinates to indicate where you have defined, referenced, or modified the entities. If you declared the entity, the coordinates are marked with a \$. If you initialized the entity, the coordinates are marked with a \*. If you both declared and initialized the entity at the same place, the coordinates are marked with a &. If the entity is set, the coordinates are marked with a @. If the entity is referenced, the coordinates are not marked.

Class is one of the following:

- Automatic
- BSS (uninitialized static internal)
- Common
- Common block
- Construct name
- Controlled (for an allocatable object)
- Controlled automatic (for an automatic object)
- Defined assignment
- Defined operator
- Derived type definition
- Entry
- External subprogram
- Function
- Generic name
- Internal subprogram
- Intrinsic
- Module
- Module function
- Module subroutine
- Namelist
- Pointee
- Private component
- Program
- Reference parameter
- Renames
- Static
- Subroutine

- Use associated
- Value parameter

If you specify the **full** suboption with **-qxref** or **-qattr**, XL Fortran reports all entities in the compilation unit. If you do not specify this suboption, only the entities you actually use appear.

---

## Object section

XL Fortran produces this section only when the **-qlist** compiler option is in effect. It contains the object code listing, which shows the source line number, the instruction offset in hexadecimal notation, the assembler mnemonic of the instruction, and the hexadecimal value of the instruction. On the right side, it also shows the cycle time of the instruction and the intermediate language of the compiler. Finally, the total cycle time (straight-line execution time) and the total number of machine instructions that are produced are displayed. There is a separate section for each compilation unit.

---

## File table section

This section contains a table that shows the file number and file name for each main source file and include file used. It also lists the line number of the main source file at which the include file is referenced. This section is always present.

---

## Compilation unit epilogue Section

This is the last section of the listing for each compilation unit. It contains the diagnostics summary and indicates whether the unit was compiled successfully. This section is not present in the listing if the file contains only one compilation unit.

---

## Compilation epilogue Section

The compilation epilogue section occurs only once at the end of the listing. At completion of the compilation, XL Fortran presents a summary of the compilation: number of source records that were read, compilation start time, compilation end time, total compilation time, total CPU time, and virtual CPU time. This section is always present in a listing.



---

## Chapter 9. Fortran-Related AIX commands

You can use some AIX commands with XL Fortran files to perform several tasks, as outlined in this section.

**Related information:** See the *AIX Commands Reference* for full details on these commands.

---

### Working with object-code archives (ar)

The **ar** command performs operations on libraries of object files that are used during linking. You might use this command to create a library of support routines that can be linked into many different programs, as follows:

```
ar -q ~/mylibs/graphics.a raytrace.o shade.o illuminate.o
xlf95 spheres.f -L~/mylibs/ -lgraphics
```

---

### Printing output files with Fortran ASA carriage controls (asa)

The **asa** command translates the output of Fortran programs that use the historical Fortran convention for ASA carriage-control characters.

The translated output is suitable for the **qprt** command:

```
generate_output | asa | qprt
```

The **fpr** command is the same as the **asa** command.

For a list of supported carriage-control characters, see *Formatted Records* in the *XL Fortran Language Reference*.

---

### Splitting subprograms into individual files (fsplit)

The **fsplit** command splits the specified Fortran source program files into several files. You should only use **fsplit** with FORTRAN 77 programs.

```
$ cat fsplit.f
 subroutine sub1
 print *, 'Hello world'
 end

 subroutine sub2
 print *, 'Goodbye'
 end

 program main
 call sub1
 call sub2
 end
$ fsplit fsplit.f
sub1.f
sub2.f
main.f
```

---

## Automating large, complex compilations (make)

The **make** command allows you to specify rules (sets of commands and options) to use for processing different types of files. By keeping track of which files are out-of-date and need to be recompiled, it can automate some or all aspects of the compilation process.

If you use **make** with XL Fortran, you may not want an object or an executable file created when the compiler encounters errors. The default setting of **-qhalt=s** prevents the compiler from generating the object file if it finds problems that it cannot correct.

**Important:** If you make any changes to the default configuration file and then move or copy your makefiles to another system, you also need to copy the changed configuration file.

---

## Run-time profiling (prof, gprof)

The **prof** and **gprof** commands provide different levels of run-time profile reports, which you can examine to find performance bottlenecks and identify the subprograms that are called most or least often. This information can help you decide where to concentrate performance-tuning efforts.

See “-p option” on page 103 for information on compiling for profiling and an example of the sequence of commands.

The “-qipa option” on page 169 allows you to feed profiling information back into subsequent compilations to enhance optimization.

---

## Translating programs into RATFOR (struct)

The **struct** command translates a FORTRAN 77 source program into a RATFOR program:

```
struct fortran.f >ratfor.f
```

---

## Displaying information inside binary files (what)

The **what** command reads information encoded into some binary files, as follows:

- Information about the compiler version is encoded in `/usr/lpp/xlf/bin/xlfentry`.
- Information about the bit mode, the compiler that created the `.mod` file, the date and time the `.mod` file was created, and the source file is encoded in each `.mod` file.

---

## Appendix A. XL Fortran technical information

This section contains details about XL Fortran that advanced programmers may need to diagnose unusual problems, run the compiler in a specialized environment, or do other things that a casual programmer is rarely concerned with.

---

### The compiler phases

The typical compiler invocation command executes some or all of the following programs in sequence. For link-time optimizations, some of the phases 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. A preprocessor
2. The compiler, which consists of the following phases:
  - a. Front-end parsing and semantic analysis
  - b. Loop transformations
  - c. Interprocedural analysis
  - d. Optimization
  - e. Register allocation
  - f. Final assembly
3. The assembler (for any `.s` files)
4. The linker `ld`

---

### External Names in the XL Fortran Shared Libraries

The run-time libraries included in the XL Fortran Run-Time Environment are AIX shared libraries, which are processed by the linker to resolve all references to external names. To minimize naming conflicts between user-defined names and the names that are defined in the run-time libraries, the names of input/output routines in the run-time libraries are prefixed with an underscore(`_`), or `_xl`.

---

### The XL Fortran run-time environment

Object code that the XL Fortran compiler produces often invokes compiler-supplied subprograms at run time to handle certain complex tasks. These subprograms are collected into several libraries.

The function of the XL Fortran Run-Time Environment may be divided into these main categories:

- Support for Fortran I/O operations
- Mathematical calculation
- Operating-system services
- Support for SMP parallelization

The XL Fortran Run-Time Environment also produces run-time diagnostic messages in the national language appropriate for your system. Unless you bind statically, you cannot run object code produced by the XL Fortran compiler without the XL Fortran Run-Time Environment.

The XL Fortran Run-Time Environment is upward-compatible. Programs that are compiled and linked with a given level of the run-time environment and a given

level of the operating system require the same or higher levels of both the run-time environment and the operating system to run.

## External names in the run-time environment

Run-time subprograms are collected into libraries. By default, the compiler invocation command also invokes the linker and gives it the names of the libraries that contain run-time subprograms called by Fortran object code.

The names of these run-time subprograms are external symbols. When object code that is produced by the XL Fortran compiler calls a run-time subprogram, the `.o` object code file contains an external symbol reference to the name of the subprogram. A library contains an external symbol definition for the subprogram. The linker resolves the run-time subprogram call with the subprogram definition.

You should avoid using names in your XL Fortran program that conflict with names of run-time subprograms. Conflict can arise under two conditions:

- The name of a subroutine, function, or common block that is defined in a Fortran program has the same name as a library subprogram.
- The Fortran program calls a subroutine or function with the same name as a library subprogram but does not supply a definition for the called subroutine or function.

---

## Technical details of the `-qfloat=hsflt` option

The `-qfloat=hsflt` option is unsafe for optimized programs that compute floating-point values that are outside the range of representation of single precision, not just outside the range of the result type. The range of representation includes both the precision and the exponent range.

Even when you follow the rules that are stated in the preceding paragraph and in “`-qfloat` option” on page 152, programs that are sensitive to precision differences might not produce expected results. Because `-qfloat=hsflt` is not compliant with IEEE, programs will not always run as expected. If a program gives unexpected, incorrect, or unacceptable results when compiled with this option, use `-qfloat=hssngl` instead.

For example, in the following program, `X.EQ.Y` may be true or may be false:

```
REAL X, Y, A(2)
DOUBLE PRECISION Z
LOGICAL SAME

READ *, Z
X = Z
Y = Z
IF (X.EQ.Y) SAME = .TRUE.
! ...
! ... Calculations that do not change X or Y
! ...
CALL SUB(X) ! X is stored in memory with truncated fraction.
IF (X.EQ.Y) THEN ! Result might be different than before.
...

A(1) = Z
X = Z
A(2) = 1. ! A(1) is stored in memory with truncated fraction.
IF (A(1).EQ.X) THEN ! Result might be different than expected.
...

```

If the value of *Z* has fractional bits that are outside the precision of a single-precision variable, these bits may be preserved in some cases and lost in others. This makes the exact results unpredictable when the double-precision value of *Z* is assigned to single-precision variables. For example, passing the variable as a dummy argument causes its value to be stored in memory with a fraction that is truncated rather than rounded.

Using `-qfloat=hsflt` can show a performance benefit for applications that perform COMPLEX division and floating-point conversions. Additionally, specifying `-qfloat=hsflt` when compiling applications targeting common, POWER, and POWER2 architectures, can achieve a broader range of performance improvements unique to those targets.

---

## Implementation details for `-qautodbl` promotion and padding

The following sections provide additional details about how the `-qautodbl` option works, to allow you to predict what happens during promotion and padding.

### Terminology

The *storage relationship* between two data objects determines the relative starting addresses and the relative sizes of the objects. The `-qautodbl` option tries to preserve this relationship as much as possible.

Data objects can also have a *value relationship*, which determines how changes to one object affect another. For example, a program might store a value into one variable, and then read the value through a different storage-associated variable. With `-qautodbl` in effect, the representation of one or both variables might be different, so the value relationship is not always preserved.

An object that is affected by this option may be:

- *Promoted*, meaning that it is converted to a higher-precision data type. Usually, the resulting object is twice as large as it would be by default. Promotion applies to constants, variables, derived-type components, arrays, and functions (which include intrinsic functions) of the appropriate types.

**Note:** `BYTE`, `INTEGER`, `LOGICAL`, and `CHARACTER` objects are never promoted.

- *Padded*, meaning that the object keeps its original type but is followed by undefined storage space. Padding applies to `BYTE`, `INTEGER`, `LOGICAL`, and nonpromoted `REAL` and `COMPLEX` objects that may share storage space with promoted items. For safety, `POINTERS`, `TARGETS`, actual and dummy arguments, members of `COMMON` blocks, structures, pointee arrays, and pointee `COMPLEX` objects are always padded appropriately depending on the `-qautodbl` suboption. This is true whether or not they share storage with promoted objects.

Space added for padding ensures that the storage-sharing relationship that existed before conversion is maintained. For example, if array elements `I(20)` and `R(10)` start at the same address by default and if the elements of `R` are promoted and become twice as large, the elements of `I` are padded so that `I(20)` and `R(10)` still start at the same address.

Except for unformatted I/O statements, which read and write any padding that is present within structures, I/O statements do not process padding.

**Note:** The compiler does not pad `CHARACTER` objects.

## Examples of storage relationships for -qautodbl suboptions

The examples in this section illustrate storage-sharing relationships between the following types of entities:

- REAL(4)
- REAL(8)
- REAL(16)
- COMPLEX(4)
- COMPLEX(8)
- COMPLEX(16)
- INTEGER(8)
- INTEGER(4)
- CHARACTER(16).

**Note:** In the diagrams, solid lines represent the actual data, and dashed lines represent padding.

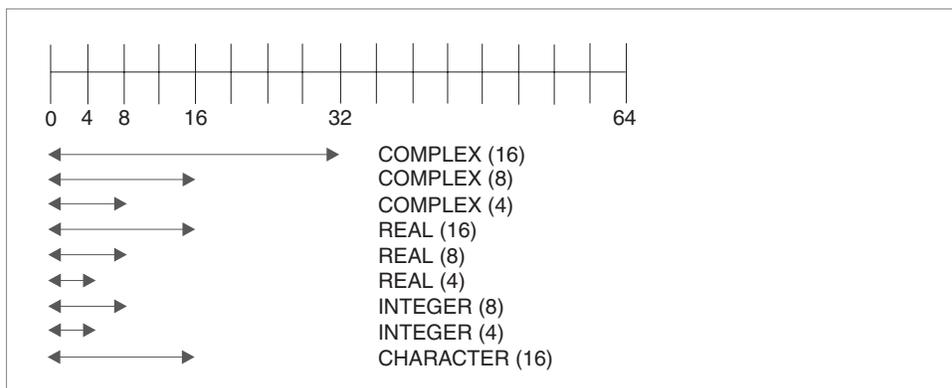


Figure 1. Storage relationships without the -qautodbl option

The figure above illustrates the default storage-sharing relationship of the compiler.

```
@process autodbl(none)
 block data
 complex(4) x8 /(1.123456789e0,2.123456789e0)/
 real(16) r16(2) /1.123q0,2.123q0/
 integer(8) i8(2) /1000,2000/
 character*5 c(2) /"abcde","12345"/
 common /named/ x8,r16,i8,c
 end

 subroutine s()
 complex(4) x8
 real(16) r16(2)
 integer(8) i8(2)
 character*5 c(2)
 common /named/ x8,r16,i8,c
 ! x8 = (1.123456e0,2.123456e0) ! promotion did not occur
 ! r16(1) = 1.123q0 ! no padding
 ! r16(2) = 2.123q0 ! no padding
 ! i8(1) = 1000 ! no padding
 ! i8(2) = 2000 ! no padding
 ! c(1) = "abcde" ! no padding
 ! c(2) = "12345" ! no padding
 end subroutine s
```

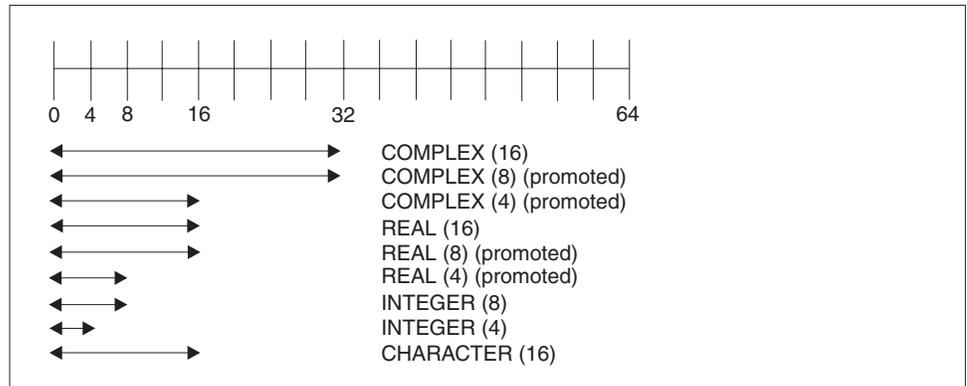


Figure 2. Storage relationships with `-qautodbl=dbl`

```

@process autodbl(db1)
 block data
 complex(4) x8
 real(16) r16(2) /1.123q0,2.123q0/
 real(8) r8
 real(4) r4 /1.123456789e0/
 integer(8) i8(2) /1000,2000/
 character*5 c(2) /"abcde","12345"/
 equivalence (x8,r8)
 common /named/ r16,i8,c,r4
! Storage relationship between r8 and x8 is preserved.
! Data values are NOT preserved between r8 and x8.
 end

 subroutine s()
 real(16) r16(2)
 real(8) r4
 integer(8) i8(2)
 character*5 c(2)
 common /named/ r16,i8,c,r4
! r16(1) = 1.123q0 ! no padding
! r16(2) = 2.123q0 ! no padding
! r4 = 1.123456789d0 ! promotion occurred
! i8(1) = 1000 ! no padding
! i8(2) = 2000 ! no padding
! c(1) = "abcde" ! no padding
! c(2) = "12345" ! no padding
 end subroutine s

```

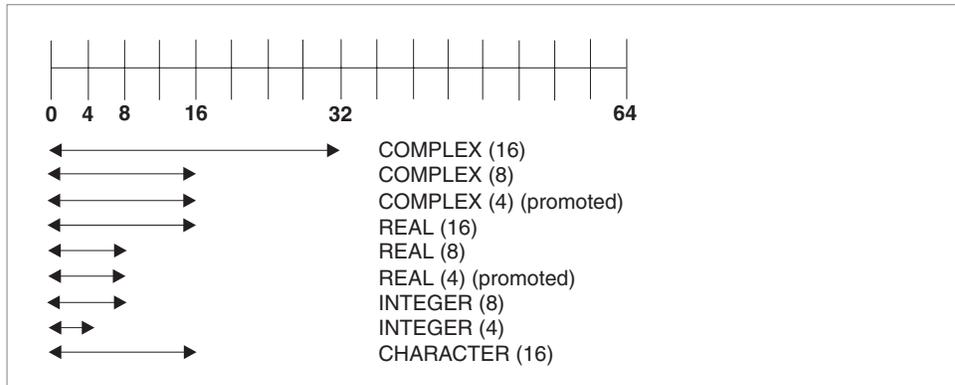


Figure 3. Storage relationships with `-qautobl=dbl4`

```
@process autodb1(db14)
 complex(8) x16 /(1.123456789d0,2.123456789d0)/
 complex(4) x8
 real(4) r4(2)
 equivalence (x16,x8,r4)
! Storage relationship between r4 and x8 is preserved.
! Data values between r4 and x8 are preserved.
! x16 = (1.123456789d0,2.123456789d0) ! promotion did not occur
! x8 = (1.123456789d0,2.123456789d0) ! promotion occurred
! r4(1) = 1.123456789d0 ! promotion occurred
! r4(2) = 2.123456789d0 ! promotion occurred
end
```

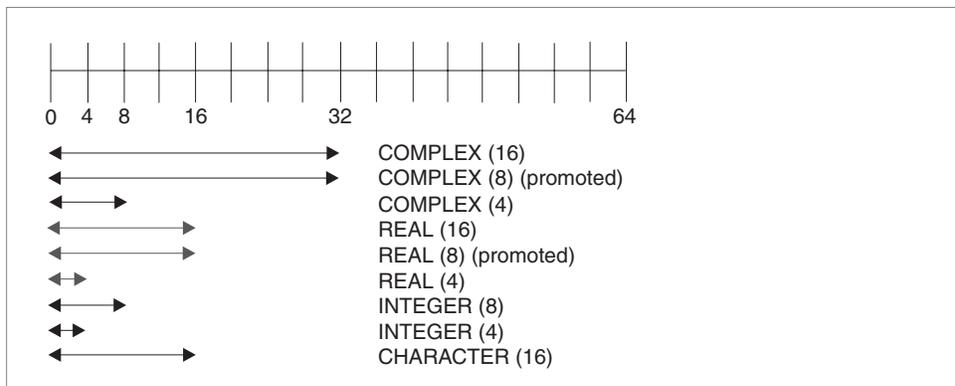


Figure 4. Storage relationships with `-qautodbl=dbl8`

```
@process autodb1(db18)
 complex(8) x16 /(1.123456789123456789d0,2.123456789123456789d0)/
 complex(4) x8
 real(8) r8(2)
 equivalence (x16,x8,r8)
! Storage relationship between r8 and x16 is preserved.
! Data values between r8 and x16 are preserved.
! x16 = (1.123456789123456789q0,2.123456789123456789q0)
!
! x8 = upper 8 bytes of r8(1) ! promotion did not occur
! r8(1) = 1.123456789123456789q0 ! promotion occurred
! r8(2) = 2.123456789123456789q0 ! promotion occurred
end
```

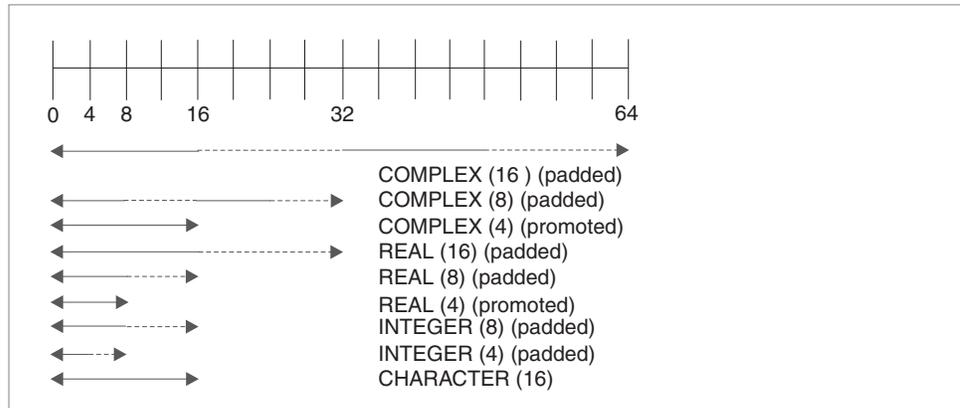


Figure 5. Storage relationships with `-qautodbl=dblpad4`

In the figure above, the dashed lines represent the padding.

```
@process autodbl(dblpad4)
 complex(8) x16 /(1.123456789d0,2.123456789d0)/
 complex(4) x8
 real(4) r4(2)
 integer(8) i8(2)
 equivalence(x16,x8,r4,i8)
! Storage relationship among all entities is preserved.
! Date values between x8 and r4 are preserved.
! x16 = (1.123456789d0,2.123456789d0) ! padding occurred
! x8 = (upper 8 bytes of x16, 8 byte pad) ! promotion occurred
! r4(1) = real(x8) ! promotion occurred
! r4(2) = imag(x8) ! promotion occurred
! i8(1) = real(x16) ! padding occurred
! i8(2) = imag(x16) ! padding occurred
end
```

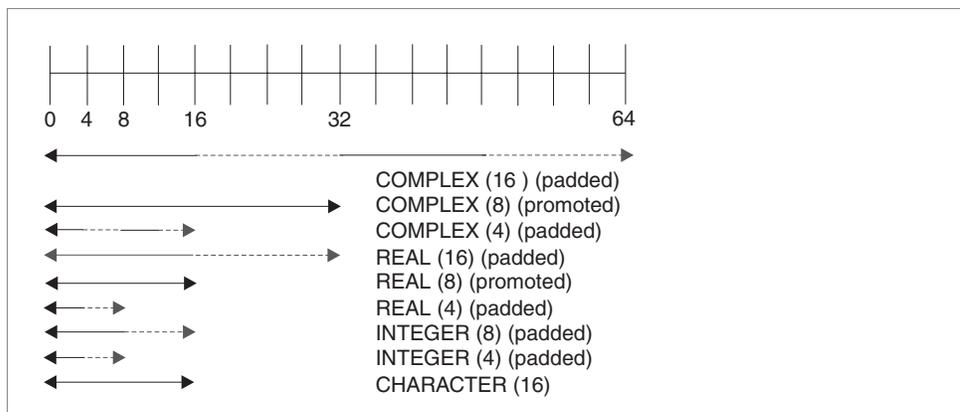


Figure 6. Storage relationships with `-qautodbl=dblpad8`

In the figure above, the dashed lines represent the padding.

```
@process autodbl(dblpad8)
 complex(8) x16 /(1.123456789123456789d0,2.123456789123456789d0)/
 complex(4) x8
 real(8) r8(2)
 integer(8) i8(2)
 byte b(16)
 equivalence (x16,x8,r8,i8,b)
! Storage relationship among all entities is preserved.
```

```

! Data values between r8 and x16 are preserved.
! Data values between i8 and b are preserved.
! x16 = (1.123456789123456789q0,2.123456789123456789q0)
! ! promotion occurred
! x8 = upper 8 bytes of r8(1) ! padding occurred
! r8(1) = real(x16) ! promotion occurred
! r8(2) = imag(x16) ! promotion occurred
! i8(1) = upper 8 bytes of real(x16) ! padding occurred
! i8(2) = upper 8 bytes of imag(x16) ! padding occurred
! b(1:8)= i8(1) ! padding occurred
! b(9:16)= i8(2) ! padding occurred
end

```

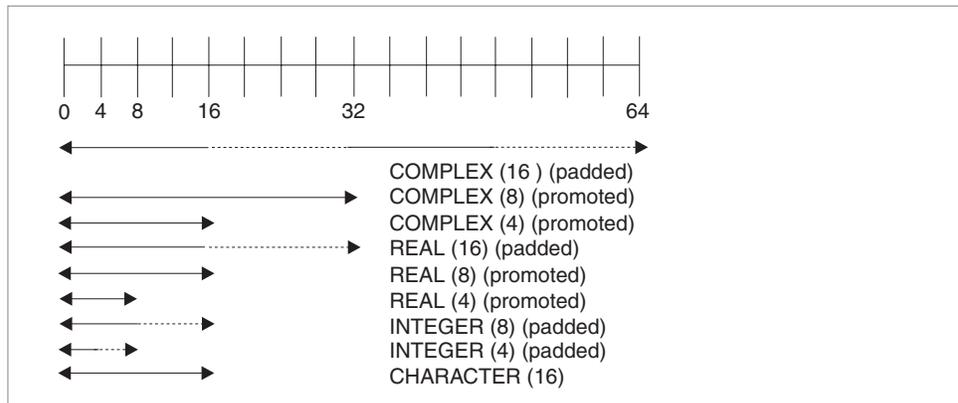


Figure 7. Storage relationships with `-qautodbl=dblpad`

In the figure above, the dashed lines represent the padding.

```

@process autodbl(db1pad)
 block data
 complex(4) x8 /(1.123456789e0,2.123456789e0)/
 real(16) r16(2) /1.123q0,2.123q0/
 integer(8) i8(2) /1000,2000/
 character*5 c(2) /"abcde","12345"/
 common /named/ x8,r16,i8,c
 end
 subroutine s()
 complex(8) x8
 real(16) r16(4)
 integer(8) i8(4)
 character*5 c(2)
 common /named/ x8,r16,i8,c
! x8 = (1.123456789d0,2.123456789d0) ! promotion occurred
! r16(1) = 1.123q0 ! padding occurred
! r16(3) = 2.123q0 ! padding occurred
! i8(1) = 1000 ! padding occurred
! i8(3) = 2000 ! padding occurred
! c(1) = "abcde" ! no padding occurred
! c(2) = "12345" ! no padding occurred
 end subroutine s

```

## Appendix B. XL Fortran internal limits

| Language Feature                                                                                                                                     | Limit                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Maximum number of iterations performed by <b>DO</b> loops with loop control with index variable of type INTEGER( <i>n</i> ) for <i>n</i> = 1, 2 or 4 | $(2^{**31})-1$                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Maximum number of iterations performed by <b>DO</b> loops with loop control with index variable of type INTEGER(8)                                   | $(2^{**63})-1$                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Maximum character format field width                                                                                                                 | $(2^{**31})-1$                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Maximum length of a format specification                                                                                                             | $(2^{**31})-1$                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Maximum length of Hollerith and character constant edit descriptors                                                                                  | $(2^{**31})-1$                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Maximum length of a fixed source form statement                                                                                                      | 34 000                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Maximum length of a free source form statement                                                                                                       | 34 000                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Maximum number of continuation lines                                                                                                                 | n/a <b>1</b>                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Maximum number of nested <b>INCLUDE</b> lines                                                                                                        | 64                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Maximum number of nested interface blocks                                                                                                            | 1 024                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Maximum number of statement numbers in a computed <b>GOTO</b>                                                                                        | 999                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Maximum number of times a format code can be repeated                                                                                                | $(2^{**31})-1$                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Allowable record numbers and record lengths for input/output files in 32-bit mode                                                                    | The record number can be up to $(2^{**63})-1$ . The maximum record length is $(2^{**31})-1$ bytes.                                                                                                                                                                                                                                                                                                                                                 |
| Allowable record numbers and record lengths for input/output files in 64-bit mode                                                                    | The record number can be up to $(2^{**63})-1$ , and the record length can be up to $(2^{**63})-1$ bytes.<br><br>However, for unformatted sequential files, you must use the <b>uwidth=64</b> run-time option for the record length to be greater than $(2^{**31})-1$ and up to $(2^{**63})-1$ . If you use the default <b>uwidth=32</b> run-time option, the maximum length of a record in an unformatted sequential file is $(2^{**31})-1$ bytes. |
| Allowable bound range of an array dimension                                                                                                          | The bound of an array dimension can be positive, negative, or zero within the range $-(2^{**31})$ to $2^{**31}-1$ in 32-bit mode, or $-(2^{**63})$ to $2^{**63}-1$ in 64-bit mode.                                                                                                                                                                                                                                                                 |
| Allowable external unit numbers                                                                                                                      | 0 to $(2^{**31})-1$ <b>2</b>                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Maximum numeric format field width                                                                                                                   | 2 000                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Maximum number of concurrent open files                                                                                                              | 2 000 <b>3</b>                                                                                                                                                                                                                                                                                                                                                                                                                                     |

**1** You can have as many continuation lines as you need to create a statement with a maximum of 34 000 bytes.

- 2** The value must be representable in an `INTEGER(4)` object, even if specified by an `INTEGER(8)` variable.
- 3** In practice, this value is somewhat lower because of files that the run-time system may open, such as the preconnected units 0, 5, and 6.

---

## 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:

IBM Corporation  
Lab Director  
IBM Canada Limited  
8200 Warden Avenue  
Markham, Ontario, Canada  
L6G 1C7

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.

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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

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.

OpenMP is a trademark of the OpenMP Architecture Review Board. Portions of this document may have been derived from the *OpenMP Application Program Interface, Version 2.5, (May 2005)* specification. Copyright 1997-2005 OpenMP Architecture Review Board.

---

## Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow the customer to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification, and tuning information is provided to help you debug your application software.

**Note:** Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

---

## Trademarks and service marks

The following terms, used in this publication, are trademarks or service marks of the International Business Machines Corporation in the United States or other countries or both:

|             |                     |                  |
|-------------|---------------------|------------------|
| AIX         | IBM                 | PowerPC          |
| PowerPC 601 | PowerPC 603         | PowerPC 604      |
| POWER       | POWER2              | POWER3           |
| POWER4      | POWER5              | POWER5+          |
| POWERserver | POWER2 Architecture | RISC System/6000 |
| RS/6000     | SAA                 | z/OS             |

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

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

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



---

## Glossary

This glossary defines terms that are commonly used in this document. It includes definitions developed by the American National Standards Institute (ANSI) and entries from the *IBM Dictionary of Computing*.

### A

**active processor.** See *online processor*.

**actual argument.** An expression, variable, procedure, or alternate return specifier that is specified in a procedure reference.

**alias.** A single piece of storage that can be accessed through more than a single name. Each name is an alias for that storage.

**alphabetic character.** A letter or other symbol, excluding digits, used in a language. Usually the uppercase and lowercase letters A through Z plus other special symbols (such as \$ and \_) allowed by a particular language.

**alphanumeric.** Pertaining to a character set that contains letters, digits, and usually other characters, such as punctuation marks and mathematical symbols.

**American National Standard Code for Information Interchange.** See *ASCII*.

**argument.** An expression that is passed to a function or subroutine. See also *actual argument*, *dummy argument*.

**argument association.** The relationship between an actual argument and a dummy argument during the invocation of a procedure.

**arithmetic constant.** A constant of type integer, real, or complex.

**arithmetic expression.** One or more arithmetic operators and arithmetic primaries, the evaluation of which produces a numeric value. An arithmetic expression can be an unsigned arithmetic constant, the name of an arithmetic constant, or a reference to an arithmetic variable, function reference, or a combination of such primaries formed by using arithmetic operators and parentheses.

**arithmetic operator.** A symbol that directs the performance of an arithmetic operation. The intrinsic arithmetic operators are:

+ addition  
- subtraction  
\* multiplication  
/ division  
\*\* exponentiation

**array.** An entity that contains an ordered group of scalar data. All objects in an array have the same data type and type parameters.

**array declarator.** The part of a statement that describes an array used in a program unit. It indicates the name of the array, the number of dimensions it contains, and the size of each dimension.

**array element.** A single data item in an array, identified by the array name and one or more subscripts. See also *subscript*.

**array name.** The name of an ordered set of data items.

**array section.** A subobject that is an array and is not a structure component.

**ASCII.** The standard code, using a coded character set consisting of 7-bit coded characters (8-bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. See also *Unicode*.

**asynchronous.** Pertaining to events that are not synchronized in time or do not occur in regular or predictable time intervals.

**assignment statement.** An executable statement that defines or redefines a variable based on the result of expression evaluation.

**associate name.** The name by which a selector of an **ASSOCIATE** construct is known within the construct.

**attribute.** A property of a data object that may be specified in a type declaration statement, attribute specification statement, or through a default setting.

**automatic parallelization.** The process by which the compiler attempts to parallelize both explicitly coded **DO** loops and **DO** loops generated by the compiler for array language.

### B

**binary constant.** A constant that is made of one or more binary digits (0 and 1).

**bind.** To relate an identifier to another object in a program; for example, to relate an identifier to a value, an address or another identifier, or to associate formal parameters and actual parameters.

**blank common.** An unnamed common block.

**block data subprogram.** A subprogram headed by a **BLOCK DATA** statement and used to initialize variables in named common blocks.

**bss storage.** Uninitialized static storage.

**busy-wait.** The state in which a thread keeps executing in a tight loop looking for more work once it has completed all of its work and there is no new work to do.

**byte constant.** A named constant that is of type byte.

**byte type.** A data type representing a one-byte storage area that can be used wherever a **LOGICAL(1)**, **CHARACTER(1)**, or **INTEGER(1)** can be used.

## C

**character constant.** A string of one or more alphabetic characters enclosed in apostrophes or double quotation marks.

**character expression.** A character object, a character-valued function reference, or a sequence of them separated by the concatenation operator, with optional parentheses.

**character operator.** A symbol that represents an operation, such as concatenation (`//`), to be performed on character data.

**character set.** All the valid characters for a programming language or for a computer system.

**character string.** A sequence of consecutive characters.

**character substring.** A contiguous portion of a character string.

**character type.** A data type that consists of alphanumeric characters. See also *data type*.

**chunk.** A subset of consecutive loop iterations.

**collating sequence.** The sequence in which the characters are ordered for the purpose of sorting, merging, comparing, and processing indexed data sequentially.

**comment.** A language construct for the inclusion of text in a program that has no effect on the execution of the program.

**common block.** A storage area that may be referred to by a calling program and one or more subprograms.

**compile.** To translate a source program into an executable program (an object program).

**compiler comment directive.** A line in source code that is not a Fortran statement but is recognized and acted on by the compiler.

**compiler directive.** Source code that controls what XL Fortran does rather than what the user program does.

**complex constant.** An ordered pair of real or integer constants separated by a comma and enclosed in parentheses. The first constant of the pair is the real part of the complex number; the second is the imaginary part.

**complex number.** A number consisting of an ordered pair of real numbers, expressible in the form  $a+bi$ , where  $a$  and  $b$  are real numbers and  $i$  squared equals  $-1$ .

**complex type.** A data type that represents the values of complex numbers. The value is expressed as an ordered pair of real data items separated by a comma and enclosed in parentheses. The first item represents the real part of the complex number; the second represents the imaginary part.

**conform.** To adhere to a prevailing standard. An executable program conforms to the Fortran 95 Standard if it uses only those forms and relationships described therein and if the executable program has an interpretation according to the Fortran 95 Standard. A program unit conforms to the Fortran 95 Standard if it can be included in an executable program in a manner that allows the executable program to be standard-conforming. A processor conforms to the standard if it executes standard-conforming programs in a manner that fulfills the interpretations prescribed in the standard.

**connected unit.** In XL Fortran, a unit that is connected to a file in one of three ways: explicitly via the **OPEN** statement to a named file, implicitly, or by preconnection.

**constant.** A data object with a value that does not change. The four classes of constants specify numbers (arithmetic), truth values (logical), character data (character), and typeless data (hexadecimal, octal, and binary). See also *variable*.

**construct.** A sequence of statements starting with a **SELECT CASE**, **DO**, **IF**, or **WHERE** statement, for example, and ending with the corresponding terminal statement.

**continuation line.** A line that continues a statement beyond its initial line.

**control statement.** A statement that is used to alter the continuous sequential invocation of statements; a

control statement may be a conditional statement, such as **IF**, or an imperative statement, such as **STOP**.

## D

**data object.** A variable, constant, or subobject of a constant.

**data striping.** Spreading data across multiple storage devices so that I/O operations can be performed in parallel for better performance. Also known as *disk striping*.

**data transfer statement.** A **READ**, **WRITE**, or **PRINT** statement.

**data type.** The properties and internal representation that characterize data and functions. The intrinsic types are integer, real, complex, logical, and character. See also *intrinsic*.

**debug line.** Allowed only for fixed source form, a line containing source code that is to be used for debugging. Debug lines are defined by a D or X in column 1. The handling of debug lines is controlled by the **-qdlines** and **-qxlines** compiler options.

**default initialization.** The initialization of an object with a value specified as part of a derived type definition.

**definable variable.** A variable whose value can be changed by the appearance of its name or designator on the left of an assignment statement.

**delimiters.** A pair of parentheses or slashes (or both) used to enclose syntactic lists.

**denormalized number.** An IEEE number with a very small absolute value and lowered precision. A denormalized number is represented by a zero exponent and a non-zero fraction.

**derived type.** A type whose data have components, each of which is either of intrinsic type or of another derived type.

**digit.** A character that represents a nonnegative integer. For example, any of the numerals from 0 through 9.

**directive.** A type of comment that provides instructions and information to the compiler.

**disk striping.** See *data striping*.

**DO loop.** A range of statements invoked repetitively by a **DO** statement.

**DO variable.** A variable, specified in a **DO** statement, that is initialized or incremented prior to each occurrence of the statement or statements within a **DO**

loop. It is used to control the number of times the statements within the range are executed.

**DOUBLE PRECISION constant.** A constant of type real with twice the precision of the default real precision.

**dummy argument.** An entity whose name appears in the parenthesized list following the procedure name in a **FUNCTION**, **SUBROUTINE**, **ENTRY**, or statement function statement.

**dynamic dimensioning.** The process of re-evaluating the bounds of an array each time the array is referenced.

**dynamic extent.** For a directive, the lexical extent of the directive and all subprograms called from within the lexical extent.

## E

**edit descriptor.** An abbreviated keyword that controls the formatting of integer, real, or complex data.

**elemental.** Pertaining to an intrinsic operation, procedure or assignment that is applied independently to elements of an array or corresponding elements of a set of conformable arrays and scalars.

**embedded blank.** A blank that is surrounded by any other characters.

**entity.** A general term for any of the following: a program unit, procedure, operator, interface block, common block, external unit, statement function, type, named variable, expression, component of a structure, named constant, statement label, construct, or namelist group.

**environment variable.** A variable that describes the operating environment of the process.

**executable program.** A program that can be executed as a self-contained procedure. It consists of a main program and, optionally, modules, subprograms and non-Fortran external procedures.

**executable statement.** A statement that causes an action to be taken by the program; for example, to perform a calculation, test conditions, or alter normal sequential execution.

**explicit initialization.** The initialization of an object with a value in a data statement initial value list, block data program unit, type declaration statement, or array constructor.

**explicit interface.** For a procedure referenced in a scoping unit, the property of being an internal procedure, module procedure, intrinsic procedure, external procedure that has an interface block, recursive

procedure reference in its own scoping unit, or dummy procedure that has an interface block.

**expression.** A sequence of operands, operators, and parentheses. It may be a variable, a constant, or a function reference, or it may represent a computation.

**extended-precision constant.** A processor approximation to the value of a real number that occupies 16 consecutive bytes of storage.

**external file.** A sequence of records on an input/output device. See also *internal file*.

**external name.** The name of a common block, subroutine, or other global procedure, which the linker uses to resolve references from one compilation unit to another.

**external procedure.** A procedure that is defined by an external subprogram or by a means other than Fortran.

## F

**field.** An area in a record used to contain a particular category of data.

**file.** A sequence of records. See also *external file*, *internal file*.

**file index.** See *i-node*.

**floating-point number.** A real number represented by a pair of distinct numerals. The real number is the product of the fractional part, one of the numerals, and a value obtained by raising the implicit floating-point base to a power indicated by the second numeral.

**format.** (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) To arrange such things as characters, fields, and lines.

**formatted data.** Data that is transferred between main storage and an input/output device according to a specified format. See also *list-directed* and *unformatted record*.

**function.** A procedure that returns the value of a single variable or an object and usually has a single exit. See also *intrinsic procedure*, *subprogram*.

## G

**generic identifier.** A lexical token that appears in an INTERFACE statement and is associated with all the procedures in an interface block.

## H

**hard limit.** A system resource limit that can only be raised or lowered by using root authority, or cannot be altered because it is inherent in the system or operating environments's implementation. See also *soft limit*.

**hexadecimal.** Pertaining to a system of numbers to the base sixteen; hexadecimal digits range from 0 (zero) through 9 (nine) and A (ten) through F (fifteen).

**hexadecimal constant.** A constant, usually starting with special characters, that contains only hexadecimal digits.

**high order transformations.** A type of optimization that restructures loops and array language.

**Hollerith constant.** A string of any characters capable of representation by XL Fortran and preceded with *nH*, where *n* is the number of characters in the string.

**host.** A main program or subprogram that contains an internal procedure is called the host of the internal procedure. A module that contains a module procedure is called the host of the module procedure.

**host association.** The process by which an internal subprogram, module subprogram, or derived-type definition accesses the entities of its host.

## I

**IPA.** Interprocedural analysis, a type of optimization that allows optimizations to be performed across procedure boundaries and across calls to procedures in separate source files.

**implicit interface.** A procedure referenced in a scoping unit other than its own is said to have an implicit interface if the procedure is an external procedure that does not have an interface block, a dummy procedure that does not have an interface block, or a statement function.

**implied DO.** An indexing specification (similar to a DO statement, but without specifying the word DO) with a list of data elements, rather than a set of statements, as its range.

**infinity.** An IEEE number (positive or negative) created by overflow or division by zero. Infinity is represented by an exponent where all the bits are 1's, and a zero fraction.

**i-node.** The internal structure that describes the individual files in the operating system. There is at least one i-node for each file. An i-node contains the node, type, owner, and location of a file. A table of i-nodes is stored near the beginning of a file system. Also known as *file index*.

**input/output (I/O).** Pertaining to either input or output, or both.

**input/output list.** A list of variables in an input or output statement specifying the data to be read or written. An output list can also contain a constant, an expression involving operators or function references, or an expression enclosed in parentheses.

**integer constant.** An optionally signed digit string that contains no decimal point.

**interface block.** A sequence of statements from an **INTERFACE** statement to its corresponding **END INTERFACE** statement.

**interface body.** A sequence of statements in an interface block from a **FUNCTION** or **SUBROUTINE** statement to its corresponding **END** statement.

**interference.** A situation in which two iterations within a **DO** loop have dependencies upon one another.

**internal file.** A sequence of records in internal storage. See also *external file*.

**interprocedural analysis.** See *IPA*.

**intrinsic.** Pertaining to types, operations, assignment statements, and procedures that are defined by Fortran language standards and can be used in any scoping unit without further definition or specification.

**intrinsic module.** A module that is provided by the compiler and is available to any program.

**intrinsic procedure.** A procedure that is provided by the compiler and is available to any program.

## K

**keyword.** (1) A statement keyword is a word that is part of the syntax of a statement (or directive) and may be used to identify the statement. (2) An argument keyword specifies the name of a dummy argument

**kind type parameter.** A parameter whose values label the available kinds of an intrinsic type.

## L

**lexical extent.** All of the code that appears directly within a directive construct.

**lexical token.** A sequence of characters with an indivisible interpretation.

**link-edit.** To create a loadable computer program by means of a linker.

**linker.** A program that resolves cross-references between separately compiled or assembled object

modules and then assigns final addresses to create a single relocatable load module. If a single object module is linked, the linker simply makes it relocatable.

**list-directed.** A predefined input/output format that depends on the type, type parameters, and values of the entities in the data list.

**literal.** A symbol or a quantity in a source program that is itself data, rather than a reference to data.

**literal constant.** A lexical token that directly represents a scalar value of intrinsic type.

**load balancing.** An optimization strategy that aims at evenly distributing the work load among processors.

**logical constant.** A constant with a value of either true or false (or T or F).

**logical operator.** A symbol that represents an operation on logical expressions:

|        |                                 |
|--------|---------------------------------|
| .NOT.  | (logical negation)              |
| .AND.  | (logical conjunction)           |
| .OR.   | (logical union)                 |
| .EQV.  | (logical equivalence)           |
| .NEQV. | (logical nonequivalence)        |
| .XOR.  | (logical exclusive disjunction) |

**loop.** A statement block that executes repeatedly.

## M

**\_main.** The default name given to a main program by the compiler if the main program was not named by the programmer.

**main program.** The first program unit to receive control when a program is run. See also *subprogram*.

**master thread.** The head process of a group of threads.

**module.** A program unit that contains or accesses definitions to be accessed by other program units.

**mutex.** A primitive object that provides mutual exclusion between threads. A mutex is used cooperatively between threads to ensure that only one of the cooperating threads is allowed to access shared data or run certain application code at a time.

## N

**NaN (not-a-number).** A symbolic entity encoded in floating-point format that does not correspond to a number. See also *quiet NaN, signalling NaN*.

**name.** A lexical token consisting of a letter followed by up to 249 alphanumeric characters (letters, digits, and underscores). Note that in FORTRAN 77, this was called a symbolic name.

**named common.** A separate, named common block consisting of variables.

**namelist group name.** The first parameter in the NAMELIST statement that names a list of names to be used in READ, WRITE, and PRINT statements.

**negative zero.** An IEEE representation where the exponent and fraction are both zero, but the sign bit is 1. Negative zero is treated as equal to positive zero.

**nest.** To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the nested subroutine) within another subroutine (the nesting subroutine).

**nonexecutable statement.** A statement that describes the characteristics of a program unit, data, editing information, or statement functions, but does not cause any action to be taken by the program.

**nonexisting file.** A file that does not physically exist on any accessible storage medium.

**normal.** A floating point number that is not denormal, infinity, or NaN.

**not-a-number.** See *NaN*.

**numeric constant.** A constant that expresses an integer, real, complex, or byte number.

**numeric storage unit.** The space occupied by a nonpointer scalar object of type default integer, default real, or default logical.

## O

**octal.** Pertaining to a system of numbers to the base eight; the octal digits range from 0 (zero) through 7 (seven).

**octal constant.** A constant that is made of octal digits.

**one-trip DO-loop.** A DO loop that is executed at least once, if reached, even if the iteration count is equal to 0. (This type of loop is from FORTRAN 66.)

**online processor.** In a multiprocessor machine, a processor that has been activated (brought online). The number of online processors is less than or equal to the number of physical processors actually installed in the machine. Also known as *active processor*.

**operator.** A specification of a particular computation involving one or two operands.

## P

**pad.** To fill unused positions in a field or character string with dummy data, usually zeros or blanks.

**paging space.** Disk storage for information that is resident in virtual memory but is not currently being accessed.

**PDF.** See *profile-directed feedback*.

**pointee array.** An explicit-shape or assumed-size array that is declared in an integer **POINTER** statement or other specification statement.

**pointer.** A variable that has the **POINTER** attribute. A pointer must not be referenced or defined unless it is pointer associated with a target. If it is an array, it does not have a shape unless it is pointer-associated.

**preconnected file.** A file that is connected to a unit at the beginning of execution of the executable program. Standard error, standard input, and standard output are preconnected files (units 0, 5 and 6, respectively).

**predefined convention.** The implied type and length specification of a data object, based on the initial character of its name when no explicit specification is given. The initial characters I through N imply type integer of length 4; the initial characters A through H, O through Z, \$, and \_ imply type real of length 4.

**present.** A dummy argument is present in an instance of a subprogram if it is associated with an actual argument and the actual argument is a dummy argument that is present in the invoking procedure or is not a dummy argument of the invoking procedure.

**primary.** The simplest form of an expression: an object, array constructor, structure constructor, function reference, or expression enclosed in parentheses.

**procedure.** A computation that may be invoked during program execution. It may be a function or a subroutine. It may be an intrinsic procedure, an external procedure, a module procedure, an internal procedure, a dummy procedure, or a statement function. A subprogram may define more than one procedure if it contains **ENTRY** statements.

**profile-directed feedback (PDF).** A type of optimization that uses information collected during application execution to improve performance of conditional branches and in frequently executed sections of code.

**program unit.** A main program or subprogram.

**pure.** An attribute of a procedure that indicates there are no side effects.

## Q

**quiet NaN.** A NaN (not-a-number) value that does not signal an exception. The intent of a quiet NaN is to propagate a NaN result through subsequent computations. See also *NaN*, *signalling NaN*.

## R

**random access.** An access method in which records can be read from, written to, or removed from a file in any order. See also *sequential access*.

**rank.** The number of dimensions of an array.

**real constant.** A string of decimal digits that expresses a real number. A real constant must contain a decimal point, a decimal exponent, or both.

**record.** A sequence of values that is treated as a whole within a file.

**relational expression.** An expression that consists of an arithmetic or character expression, followed by a relational operator, followed by another arithmetic or character expression.

**relational operator.** The words or symbols used to express a relational condition or a relational expression:

|      |                          |
|------|--------------------------|
| .GT. | greater than             |
| .GE. | greater than or equal to |
| .LT. | less than                |
| .LE. | less than or equal to    |
| .EQ. | equal to                 |
| .NE. | not equal to             |

**result variable.** The variable that returns the value of a function.

**return specifier.** An argument specified for a statement, such as *CALL*, that indicates to which statement label control should return, depending on the action specified by the subroutine in the *RETURN* statement.

## S

**scalar.** (1) A single datum that is not an array. (2) Not having the property of being an array.

**scale factor.** A number indicating the location of the decimal point in a real number (and, on input, if there is no exponent, the magnitude of the number).

**scope.** That part of an executable program within which a lexical token has a single interpretation.

**scope attribute.** That part of an executable program within which a lexical token has a single interpretation of a particular named property or entity.

**scoping unit.** (1) A derived-type definition. (2) An interface body, excluding any derived-type definitions and interface bodies contained within it. (3) A program unit or subprogram, excluding derived-type definitions, interface bodies, and subprograms contained within it.

**selector.** The object that is associated with the associate name in an *ASSOCIATE* construct.

**semantics.** The relationships of characters or groups of characters to their meanings, independent of the manner of their interpretation and use. See also *syntax*.

**sequential access.** An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file. See also *random access*.

**signaling NaN.** A NaN (not-a-number) value that signals an invalid operation exception whenever it appears as an operand. The intent of the signaling NaN is to catch program errors, such as using an uninitialized variable. See also *NaN*, *quiet NaN*.

**sleep.** The state in which a thread completely suspends execution until another thread signals it that there is work to do.

**SMP.** See *symmetric multiprocessing*.

**soft limit.** A system resource limit that is currently in effect for a process. The value of a soft limit can be raised or lowered by a process, without requiring root authority. The soft limit for a resource cannot be raised above the setting of the hard limit. See also *hard limit*.

**spill space.** The stack space reserved in each subprogram in case there are too many variables to hold in registers and the program needs temporary storage for register contents.

**specification statement.** A statement that provides information about the data used in the source program. The statement could also supply information to allocate data storage.

**stanza.** A group of lines in a file that together have a common function or define a part of the system. Stanzas are usually separated by blank lines or colons, and each stanza has a name.

**statement.** A language construct that represents a step in a sequence of actions or a set of declarations. Statements fall into two broad classes: executable and nonexecutable.

**statement function.** A name, followed by a list of dummy arguments, that is equated with an intrinsic or derived-type expression, and that can be used as a substitute for the expression throughout the program.

**statement label.** A number from one through five digits that is used to identify a statement. Statement

labels can be used to transfer control, to define the range of a **DO**, or to refer to a **FORMAT** statement.

**storage association.** The relationship between two storage sequences if a storage unit of one is the same as a storage unit of the other.

**structure.** A scalar data object of derived type.

**structure component.** The part of a data object of derived-type corresponding to a component of its type.

**subobject.** A portion of a named data object that may be referenced or defined independently of other portions. It can be an array element, array section, structure component, or substring.

**subprogram.** A function subprogram or a subroutine subprogram. Note that in FORTRAN 77, a block data program unit was called a subprogram. See also *main program*.

**subroutine.** A procedure that is invoked by a **CALL** statement or defined assignment statement.

**subscript.** A subscript quantity or set of subscript quantities enclosed in parentheses and used with an array name to identify a particular array element.

**substring.** A contiguous portion of a scalar character string. (Although an array section can specify a substring selector, the result is not a substring.)

**symmetric multiprocessing (SMP).** A system in which functionally-identical multiple processors are used in parallel, providing simple and efficient load-balancing.

**synchronous.** Pertaining to an operation that occurs regularly or predictably with regard to the occurrence of a specified event in another process.

**syntax.** The rules for the construction of a statement. See also *semantics*.

## T

**target.** A named data object specified to have the **TARGET** attribute, a data object created by an **ALLOCATE** statement for a pointer, or a subobject of such an object.

**thread.** A stream of computer instructions that is in control of a process. A multithread process begins with one stream of instructions (one thread) and may later create other instruction streams to perform tasks.

**thread visible variable.** A variable that can be accessed by more than one thread.

**time slice.** An interval of time on the processing unit allocated for use in performing a task. After the interval has expired, processing unit time is allocated to

another task, so a task cannot monopolize processing unit time beyond a fixed limit.

**token.** In a programming language, a character string, in a particular format, that has some defined significance.

**trigger constant.** A sequence of characters that identifies comment lines as compiler comment directives.

**type declaration statement.** A statement that specifies the type, length, and attributes of an object or function. Objects can be assigned initial values.

## U

**unformatted record.** A record that is transmitted unchanged between internal and external storage.

**Unicode.** A universal character encoding standard that supports the interchange, processing, and display of text that is written in any of the languages of the modern world. It also supports many classical and historical texts in a number of languages. The Unicode standard has a 16-bit international character set defined by ISO 10646. See also *ASCII*.

**unit.** A means of referring to a file to use in input/output statements. A unit can be connected or not connected to a file. If connected, it refers to a file. The connection is symmetric: that is, if a unit is connected to a file, the file is connected to the unit.

**unsafe option.** Any option that could result in undesirable results if used in the incorrect context. Other options may result in very small variations from the default result, which is usually acceptable. Typically, using an unsafe option is an assertion that your code is not subject to the conditions that make the option unsafe.

**use association.** The association of names in different scoping units specified by a **USE** statement.

## V

**variable.** A data object whose value can be defined and redefined during the execution of an executable program. It may be a named data object, array element, array section, structure component, or substring. Note that in FORTRAN 77, a variable was always scalar and named.

## X

**XPG4.** X/Open Common Applications Environment (CAE) Portability Guide Issue 4; a document which defines the interfaces of the X/Open Common Applications Environment that is a superset of

POSIX.1-1990, POSIX.2-1992, and POSIX.2a-1992  
containing extensions to POSIX standards from XPG3.

## Z

**zero-length character.** A character object that has a length of 0 and is always defined.

**zero-sized array.** An array that has a lower bound that is greater than its corresponding upper bound. The array is always defined.



# Index

## Special characters

`_OPENMP` C preprocessor macro 29, 223

`#` compiler option 77

`-l` compiler option 78

`-B` compiler option 79

`-b64` linker option 80

`-bdynamic` linker option 81

`-bhalt` linker option 83

`-bloadmap` linker option 84

`-bmap` linker option 72

`-bmaxdata` and `-bmaxstack` linker options 85

`-brename` linker option 72

`-brtl` linker option 86

`-bshared` linker option 81

`-bstatic` linker option 81

`-c` compiler option 90

`-C` compiler option 89

`-d` compiler option 92

`-D` compiler option 91

`-F` compiler option 93

`-g` compiler option 94, 282

`-I` compiler option 95

`-k` compiler option 96

`-l` compiler option 98

`-L` compiler option 97

`-NS` compiler option 99

`-o` compiler option 102

`-O` compiler option 100

`-O2` compiler option 100

`-O3` compiler option 100

`-O4` compiler option 100

`-O5` compiler option 101

`-p` compiler option 103

`-Q`, `-Q!`, `-Q+`, `-Q-` compiler options 104

`-q32` compiler option 105

`-q64` compiler option 106

`-qalias` compiler option 109

`-qalign` compiler option 112

`-qarch` compiler option 28, 114

`-qassert` compiler option 119

`-qattr` compiler option 120, 298

`-qautodbl` compiler option 121, 305

`-qcache` compiler option 28, 124

`-qclines` compiler option 126

`-qcheck` compiler option 89, 127

`-qci` compiler option 128

`-qcompact` compiler option 129

`-qcr` compiler option 130

`-qctyp1ss` compiler option 131

`-qdbg` compiler option 94, 133

`-qddim` compiler option 134

`-qdirective` compiler option 135

`-qdlines` compiler option 91, 138

`-qdpc` compiler option 139

`-qdpcl` compiler option 140

`-qenablevmx` compiler option 141

`-qenum` compiler option 142

`-qescape` compiler option 143

`-qessl` compiler option 144

`-qextchk` compiler option 145

`-qextern` compiler option 146

`-qextname` compiler option 147

`-qfdpr` compiler option 149

`-qfixed` compiler option 150

`-qflag` compiler option 151

`-qfloat` compiler option 152

`-qfltrap` compiler option 154

`-qfree` compiler option 156

`-qfullpath` compiler option 157

`-qhalt` compiler option 158

`-qhot` compiler option 159

`-qieee` compiler option 162, 269

`-qinit` compiler option 163

`-qinitauto` compiler option 164

`-qintlog` compiler option 166

`-qintsize` compiler option 167

`-qipa` compiler option 169

`-qkeepparm` compiler option 176

`-qlanglvl` compiler option 177

`-qlargepage` compiler option 179

`-qlibansi` compiler option 180

`-qlibansi` linker option 173

`-qlibessl` linker option 173

`-qlibposix` linker option 173

`-qlinedebug` compiler option 183

`-qlist` compiler option 184, 299

`-qlistopt` compiler option 185, 295

`-qlog4` compiler option 186

`-qmaxmem` compiler option 187

`-qmbcs` compiler option 189

`-qmixed` compiler option 190

`-qmoddir` compiler option 191

`-qmodule` compiler option 192

`-qnoprint` compiler option 193

`-qnulterm` compiler option 194

`-qobject` compiler option 195

`-qonetrip` compiler option 78, 196

`-qoptimize` compiler option 100, 197

`-qpdf` compiler option 198

`-qphsinfo` compiler option 202

`-qpic` compiler option 204

`-qport` compiler option 205

`-qposition` compiler option 207

`-qprefetch` compiler option 208

`-qqcount` compiler option 209

`-qrealize` compiler option 210

`-qrecur` compiler option 212

`-qreport` compiler option 213, 297

`-qsaa` compiler option 215

`-qsave` compiler option 216

`-qsaveopt` compiler option 217

`-qscl` compiler option 218

`-qshowpdf` compiler option 219

`-qsigtrap` compiler option 220

`-qsmallstack` compiler option 221

`-qsm` compiler option 222

`-qsource` compiler option 227, 296

`-qspills` compiler option 99, 228

`-qstacktemp` compiler option 229

`-qstrict` compiler option 230

`-qstrict_induction` compiler option 232

`-qstrictieemod` compiler option 231

`-qsuffix` compiler option 233

`-qsuppress` compiler option 234

`-qswapomp` compiler option 236

`-qtbtable` compiler option 238

`-qthreaded` compiler option 239

`-qtune` compiler option 28, 240

`-qundef` compiler option 243, 263

`-qunroll` compiler option 244

`-qunwind` compiler option 245

`-qvecnv` compiler option 246

`-qversion` compiler option 247

`-qwarn64` compiler option 248

`-qxflag=dvz` compiler option 249

`-qxflag=oldtab` compiler option 250

`-qxflag=xalias` compiler option (obsolete)

*See* `-qalias` compiler option

`-qxf77` compiler option 252

`-qxf90` compiler option 254

`-qxlines` compiler option 256

`-qxref` compiler option 258, 298

`-qzerosize` compiler option 259

`-S` compiler option 260

`-t` compiler option 261

`-u` compiler option 263

`-U` compiler option 262

`-v` compiler option 264

`-V` compiler option 265

`-w` compiler option 151, 268

`-W` compiler option 266

`-yn`, `-ym`, `-yp`, `-yz` compiler options 162, 269

`/etc/csh.cshrc` and `/etc/csh.login` files 8

`/etc/xf.cfg` configuration file 11, 93

`/tmp` directory

*See* `TMPDIR` environment variable

`/usr/lib/lib*.a` library files 23, 31, 32, 33, 34

`/usr/lpp/xf/bin/xfentry` file 302

`/usr/lpp/xf/include` directory 39

`/usr/lpp/xf/include_d7` directory 39

`/usr/lpp/xf/lib/lib*.a` library files 23, 31, 32, 33, 34

`.a` files 23

`.cfg` files 23

`.cshrc` file 8

`.f` and `.F` files 23

`.f90` suffix, compiling files with 12

`.lst` files 25

`.mod` file names 192

`.mod` files 23, 25, 39, 191, 302

`.o` files 23, 25

`.profile` file 8

`.s` files 23, 25

`.XOR` operator 252

`@PROCESS` compiler directive 27

`#if` and other `cpp` directives 30

# Numerics

1501-224,1501-229, and 1517-011 error messages 280

15xx identifiers for XL Fortran messages 276

4K suboption of -qalign 112

601 suboption of -qarch 114, 115

601 suboption of -qtune 240

603 suboption of -qarch 114, 115

603 suboption of -qtune 240

604 suboption of -qarch 114, 115

604 suboption of -qtune 240

64-bit environment 271

64-bit large data type support 271

64-bit thread support 272

## A

a.out file 25

actual arguments

definition of 317

addresses of arguments, saving 252

affinity suboption of

-qsmp=schedule 223

ALIAS @PROCESS directive 109

ALIGN @PROCESS directive 112

alignment of BIND(C) derived types 112

alignment of CSECTs and large arrays for

data-stripped I/O 112

allocatable arrays, automatic deallocation

with -qxl90=autodealloc 254

alphabetic character, definition of 317

alphanumeric, definition of 317

ANSI

checking conformance to the Fortran

90 standard 4, 44, 177

checking conformance to the Fortran

95 standard 4, 44, 177

appendold and appendunknown

suboptions of -qposition 207

ar command 301

archive files 23

argument addresses, saving 252

argument promotion (integer only) for

intrinsic procedures 252

arguments

definition of 317

passing null-terminated strings to C

functions 194

arrays

optimizing assignments 109

arrays, initialization problems 280

aryovrlp suboption of -qalias 109

as and asopt attributes of configuration file 12

as command, passing command-line

options to 28

asa command 301

ASCII

definition of 317

assembler

source (.s) files 23, 25

ATTR @PROCESS directive 120

attribute section in compiler listing 298

auto suboption of -qarch 114

auto suboption of -qsmp 222

auto suboption of -qtune 240

AUTODBL @PROCESS directive 121

autodealloc suboption of -qxl90 254

## B

big data and stack segments 85

binary compatibility, POSIX pthreads 38

BIND(C) derived types, alignment 112

blankpad suboption of -qxl77 252

bolt attribute of configuration file 12

bss storage, alignment of arrays in 112

buffering run-time option

description 41

using with preconnected files 41

## C

C preprocessor (cpp) 29

carriage control characters, printing files

with 301

carriage return character 130

CCLINES @PROCESS 126

character constants and typeless

constants 131

CHECK @PROCESS directive 89, 127

chunk

definition of 318

CI @PROCESS directive 128

cleanpdf command 199

clonearch suboption of -qipa 170

cloneproc suboption of -qipa 170

cnvrr run-time option 43

code attribute of configuration file 12

code generation for different systems 28

code optimization 5

column 1 and carriage control

characters 301

com suboption of -qarch 114

command line, specifying options on 26

command-line options

See compiler options

common blocks, finding sizes of 72

COMPACT @PROCESS directive 129

compilation order 23

compilation unit epilogue section in

compiler listing 299

compile-time suboptions, -qipa 169

compiler listings 295

See also listings

compiler options for controlling 62

compiler options

See also the individual options listed

under Special Characters at the start of the index

deprecated 74

descriptions 76

for compatibility 64

for controlling input to the

compiler 52

for controlling listings and

messages 62

for controlling the compiler internal

operation 73

for debugging and error checking 59

for floating-point processing 71

compiler options (*continued*)

for linking 72

for performance optimization 54

obsolete or not recommended 74

scope and precedence 26

section in compiler listing 295

specifying in the source file 27

specifying on the command line 26

specifying the locations of output files 54

summary 51

compiler options for 64-bit 272

compiling

cancelling a compilation 23

description of how to compile a

program 20

problems 279

SMP programs 22

conditional compilation 29

configuration file 11, 23, 93

conflicting options

-C interferes with -qhot 89

-qautodbl overrides -qrealsize 123

-qdpcc is overridden by -qautodbl and

-qrealsize 210

-qflag overrides -qlanglvl and

-qsa 151

-qhalt is overridden by

-qnoobject 195

-qhalt overrides -qobject 195

-qhot is overridden by -C 159

-qintsize overrides -qlog4 186

-qlanglvl is overridden by -qflag 178

-qlog4 is overridden by -qintsize 186

-qnoobject overrides -qhalt 158

-qobject is overridden by -qhalt 158

-qrealsize is overridden by

-qautodbl 123, 211

-qrealsize overrides -qdpcc 210

-qsa is overridden by -qflag 215

@PROCESS overrides command-line

setting 26

command-line overrides configuration

file setting 26

specified more than once, last one

takes effect 26

conformance checking 4, 177, 215

conversion errors 43

core file 282

could not load program (error message) 279

cpp command 29

cpp, cppoptions, and cppsuffix attributes

of configuration file 12

cpu\_time\_type run-time option 43

cross-reference section in compiler

listing 298

crt attribute of configuration file 12

crt\_64 attribute of configuration file 12

CSECTs, alignment of 112

csh shell 8

cshrc, csh.cshrc, and csh.login files 8

CTYPLSS @PROCESS directive 131

customizing configuration file (including

default compiler options) 11

## D

data limit 279  
data segment, increasing size of 85  
data striping  
    -qalign required for improved performance 112  
DBG @PROCESS directive 94, 133  
dbl, dbl4, dbl8, dblpad, dblpad4, dblpad8  
    suboptions of -qautodbl 121  
dbx debugger 5  
dbx support  
    sample session 282  
DDIM @PROCESS directive 134  
debugger support 5  
debugging 275  
    compiler options for 59  
    using path names of original files 157  
default\_recl run-time option 43  
defaultmsg attribute of configuration file 12  
defaults  
    customizing compiler defaults 11  
    search paths for include and .mod files 95  
    search paths for libraries 10  
deprecated compiler options 74  
deps suboption of -qassert 119  
DIRECTIVE @PROCESS directive 135  
disassembly listing  
    from the -S compiler option 260  
disk space, running out of 280  
disk striping  
    See data striping  
DLINES @PROCESS directive 91, 138  
DPC @PROCESS directive 139  
DPCL @PROCESS directive 140  
dummy argument  
    definition of 319  
dynamic dimensioning of arrays 134  
dynamic extent, definition of 319  
dynamic linking 36  
dynamic suboption of  
    -qsmpt=schedule 224

## E

E error severity 275  
edit descriptors (B, O, Z), differences  
    between F77 and F90 252  
edit descriptors (G), difference between  
    F77 and F90 252  
editing source files 19  
emacs text editor 19  
enable suboption of -qfltrap 154  
end-of-file, writing past 252  
ENTRY statements, compatibility with  
    previous compiler versions 252  
environment problems 279  
environment variables  
    64-bit environment  
        OBJECT\_MODE 272  
    compile time 8  
        LANG 9  
        LIBPATH 10  
        NLSPATH 9

environment variables (*continued*)  
    compile time (*continued*)  
        OBJECT\_MODE 272  
        PDFDIR 10  
        TMPDIR 10  
    run-time  
        LIBPATH 49  
        PDFDIR 10  
        TMPDIR 49  
        XLFRTOPTS 40  
        XL\_NOCLONEARCH 49  
        XLFSCRATCH\_unit 11  
        XLFUNIT\_unit 11  
eof, writing past 252  
epilogue sections in compiler listing 299  
err\_recovery run-time option 44  
error checking, compiler options for 59  
error messages 275  
    1501-224 280  
    1501-229 280  
    1517-011 280  
    compiler options for controlling 62  
    explanation of format 276  
    in compiler listing 296  
erroneof run-time option 44  
ESCAPE @PROCESS directive 143  
example programs  
    See sample programs  
exception handling 49  
    for floating point 154  
exclusive or operator 252  
executable files 25  
executing a program 37  
executing the compiler 20  
exits suboption of -qipa 170  
EXTCHK @PROCESS directive 145  
external names  
    in the run-time environment 304  
EXTNAME @PROCESS directive 147

## F

f77 command  
    description 20  
    level of Fortran standard  
        compliance 15, 21  
f90 suffix 12  
file table section in compiler listing 299  
files  
    editing source 19  
    input 23  
    output 25  
    using suffixes other than .f for source files 12  
FIPS FORTRAN standard, checking  
    conformance to 4  
FIXED @PROCESS directive 150  
FLAG @PROCESS directive 151  
FLOAT @PROCESS directive 152  
floating-point  
    exception handling 49  
    exceptions 154  
fltint suboption of -qfloat 152  
FLTRAP @PROCESS directive 154  
fold suboption of -qfloat 152  
fort77 command  
    description 20

fort77 command (*continued*)  
    level of Fortran standard  
        compliance 15  
Fortran 2003 features 44  
Fortran 2003 iostat\_end behavior 44  
Fortran 90  
    compiling programs written for 21  
fppv and fppk attributes of configuration  
    file 12  
fpr command 301  
FREE @PROCESS directive 156  
fsplit command 301  
fsuffix attribute of configuration file 12  
full suboption of -qbttable 238  
FULLPATH @PROCESS directive 157

## G

G edit descriptor, difference between F77  
    and F90 252  
gcr attribute of configuration file 12  
gcr\_64 attribute of configuration file 12  
gedit77 suboption of -qxl77 252  
generating code for different systems 28  
gmon.out file 302  
gprof command 302  
guided suboption of  
    -qsmpt=schedule 224

## H

HALT @PROCESS directive 158  
hardware, compiling for different types  
    of 28  
header section in compiler listing 295  
heap storage, increasing size of 85  
hexint and nohexint suboptions of  
    -qport 205  
hot attribute of configuration file 12  
hotlist suboption of -qreport 213  
hsflt suboption of -qfloat 152, 304  
hssngl suboption of -qfloat 153  
huge data and stack segments 85

## I

I error severity 275  
i-node 46  
I/O  
    See input/output  
IEEE @PROCESS directive 162, 269  
imprecise suboption of -qfltrap 154  
include\_32 attribute of configuration  
    file 12  
include\_64 attribute of configuration  
    file 12  
inexact suboption of -qfltrap 154  
informational message 275  
INIT @PROCESS directive 163  
initialize arrays, problems 280  
inline suboption of -qipa 170  
inlining 104  
input files 23  
input/output  
    increasing throughput with data  
        striping 112

input/output (*continued*)  
 run-time behavior 40  
 when unit is positioned at  
 end-of-file 252  
 installation problems 279  
 installing the compiler 7  
 intarg suboption of `-qxl77` 252  
 integer arguments of different kinds to  
 intrinsic procedures 252  
 interface errors, detecting 34  
 internal limits for the compiler 311  
 interprocedural analysis (IPA) 169  
 INTLOG @PROCESS directive 166  
 intptr suboption of `-qalias` 109  
 intrinsic procedures accepting integer  
 arguments of different kinds 252  
 intrinths run-time option 44  
 INTSIZE @PROCESS directive 167  
 intxor suboption of `-qxl77` 252  
 invalid suboption of `-qfltrap` 154  
 invoking a program 37  
 invoking the compiler 20  
 iostat\_end run-time option 44  
 ipa attribute of configuration file 12  
 irand routine, naming restriction for 36  
 ISO  
 checking conformance to the Fortran  
 2003 standard 4  
 checking conformance to the Fortran  
 90 standard 4, 44, 177  
 checking conformance to the Fortran  
 95 standard 4, 44, 177  
 isolated suboption of `-qipa` 171  
 itercnt suboption of `-qassert` 119

## K

killed (error message) 279  
 kind type parameters 15, 167, 210  
 ksh shell 8

## L

L error severity 275  
 LANG environment variable 9  
 LANGLVL @PROCESS directive 177  
 langlvl run-time option 44  
 language support 3  
 language-level error 275  
 large and small suboptions of `-qpic` 204  
 large data and stack segments 85  
 large pages 179  
 LC\_\* national language categories 9  
 ld and ldopt attributes of configuration  
 file 12  
 ld command  
 passing command-line options to 28  
 used for linking 64-bit non-SMP  
 files 34  
 used for linking 64-bit SMP files 32  
 used for linking non-SMP files 33  
 used for linking SMP files 31  
 leadzero suboption of `-qxl77` 252  
 level of XL Fortran, determining 14  
 level suboption of `-qipa` 171  
 lexical extent, definition of 321

lib\*.a library files 23, 98  
 LIBPATH environment variable 49, 279  
 compile time 10  
 libraries 23, 31, 32, 33, 34  
 default search paths 10  
 shared 303  
 libraries attribute of configuration  
 file 12  
 library path environment variable 279  
 libxlf.a library 15  
 libxlf90\_t.a library 21  
 libxlf90.a and libxlf.a libraries 15  
 libxlf90.a library 15, 39  
 libxlfpthrds\_compat.a library 39  
 libxlsmp.a library 39  
 limit command 279  
 limits internal to the compiler 311  
 line feed character 130  
 LINEDEBUG @PROCESS directive 183  
 link-time suboptions, `-qipa` 170  
 linker options 72  
 -b64 80  
 -bdynamic 81  
 -bmaxdata 85  
 -bmaxstack 85  
 -bnortl 86  
 -brtl 86  
 -bshared 81  
 -bstatic 81  
 -qlibansi 173  
 -qlibessl 173  
 -qlibposix 173  
 linking 31  
 dynamic 36  
 problems 280  
 static 36  
 LIST @PROCESS directive 184  
 list suboption of `-qipa` 171  
 listing files 25  
 listing options 62  
 LISTOPT @PROCESS directive 185  
 locale, setting at run time 40  
 LOG4 @PROCESS directive 186  
 lowfreq suboption of `-qipa` 172  
 lspp command 14

## M

m suboption of `-y` 269  
 machines, compiling for different  
 types 28, 114  
 macro expansion 29  
 macro, `_OPENMP` C preprocessor 29,  
 223  
 maf suboption of `-qfloat` 153, 230  
 make command 77, 302  
 makefiles  
 configuration file as alternative for  
 default options 11  
 copying modified configuration files  
 along with 11  
 malloc system routine 122  
 MAXMEM @PROCESS directive 187  
 MBCS @PROCESS directive 189  
 mclock routine, naming restrictions  
 for 36  
 mcrt attribute of configuration file 12

mcrt\_64 attribute of configuration file 12  
 message suppression 234  
 messages  
 1501-053 error message 280  
 1501-224 error message 280  
 1501-229 error message 280  
 1517-011 error message 280  
 catalog files for 278  
 compiler options for controlling 62  
 copying message catalogs to another  
 system 278  
 selecting the language for run-time  
 messages 40  
 messaging  
 XL Fortran programs calling MPI  
 library 12  
 migrating 4  
 from previous versions of XL  
 Fortran 15  
 minus suboption of `-qieee` 162  
 missing suboption of `-qipa` 172  
 MIXED @PROCESS directive 190, 262  
 mod and nomod suboptions of  
 -qport 205  
 mod file names, intrinsic 192  
 mod files 23, 25, 191, 302  
 modules, effect on compilation order 23  
 mon.out file 23, 302  
 MPI library 12  
 mpix stanza of configuration file 12  
 mpix\_r stanza of configuration file 12  
 mpix\_r7 stanza of configuration file 12  
 mpix90 stanza of configuration file 12  
 mpix90\_r stanza of configuration  
 file 12  
 mpix95 stanza of configuration file 12  
 mpix95\_r stanza of configuration file 12  
 mpix95\_r7 stanza of configuration  
 file 12  
 multconn run-time option 46  
 multconnio run-time option 46

## N

n suboption of `-y` 269  
 name conflicts, avoiding 36  
 namelist run-time option 47  
 NaN values  
 specifying with `-qinitauto` compiler  
 option 164  
 nans suboption of `-qfloat` 153  
 national language support  
 at run time 40  
 compile time environment 9  
 nearest suboption of `-qieee` 162  
 nested\_par suboption of `-qsm` 222  
 network file system (NFS)  
 using the compiler on a 7  
 Network Install Manager 7  
 NFS  
 See network file system  
 NIM (Network Install Manager) 7  
 NLSPATH environment variable  
 compile time 9  
 nlwidth run-time option 47  
 noauto suboption of `-qsm` 222

nodblpad suboption of -qautodbl  
*See* none suboption instead  
 nodeps suboption of -qassert 119  
 noinline suboption of -qipa 172  
 none suboption of -qautodbl 121  
 none suboption of -qtbtable 238  
 nonested\_par suboption of -qsmp 222  
 noomp suboption of -qsmp 222  
 noopt suboption of -qsmp 223  
 norec\_locks suboption of -qsmp 223  
 null-terminated strings, passing to C  
 functions 194  
 NULLTERM @PROCESS directive 194

## O

OBJECT @PROCESS directive 195  
 object files 23, 25  
 object suboption of -qipa 169  
 OBJECT\_MODE environment  
 variable 272  
 obsolete compiler options 74  
 oldboz suboption of -qxl77 252  
 omp suboption of -qsmp 222  
 ONETRIP @PROCESS directive 78, 196  
 opt suboption of -qsmp 223  
 optimization 5  
   compiler options for 54  
 OPTIMIZE @PROCESS directive 100,  
 197  
 options attribute of configuration file 12  
 options section in compiler listing 295  
 osuffix attribute of configuration file 12  
 output files 25  
 overflow suboption of -qfltrap 154

## P

p suboption of -y 269  
 p2sc suboption of -qarch 115  
 p2sc suboption of -qtune 240  
 pad setting, changing for internal,  
 direct-access and stream-access  
 files 252  
 padding of data types with -qautodbl  
 option 305  
 paging space  
   running out of 280  
 parameters  
   *See* arguments  
 partition suboption of -qipa 172  
 path name of source files, preserving  
 with -qfullpath 157  
 PDFDIR environment variable 10  
 pdfname suboption of -qipa 172  
 performance of real operations, speeding  
 up 122, 210  
 Performance Toolbox 238  
 persistent suboption of -qxl77 252  
 PHSINFO @PROCESS directive 202  
 platform, compiling for a specific  
 type 114  
 plus suboption of -qiee 162  
 pointers (Fortran 90) and -qinit compiler  
 option 163  
 PORT @PROCESS directive 205

POSITION @PROCESS directive 207  
 POSIX pthreads  
   API support 22  
   binary compatibility 38  
   run-time libraries 39  
 POWER, POWER2, POWER3, POWER4,  
 POWER5, or PowerPC systems  
   compiling programs for 28  
 POWER, POWER2, POWER3, POWER4,  
 POWER5, POWER5+, or PowerPC  
 systems 114  
 ppc suboption of -qarch 114, 115  
 ppc64 suboption of -qarch 115  
 ppc64gr suboption of -qarch 115  
 ppc64grsq suboption of -qarch 115  
 ppc64v suboption of -qarch 114  
 ppc970 suboption of -qarch 114  
 ppc970 suboption of -qtune 240  
 ppcgr suboption of -qarch 115  
 precision of real data types 122, 210  
 preprocessing Fortran source with the C  
 preprocessor 29  
 problem determination 275  
 procedure calls to other languages  
   *See* subprograms in other languages,  
   calling  
 prof command 24, 302  
 profile file 8  
 profiling data files 24  
 proflibs attribute of configuration file 12  
 Program Editor 19  
 promoting integer arguments to intrinsic  
 procedures 252  
 promotion of data types with -qautodbl  
 option 305  
 pteovrlp suboption of -qalias 109  
 pure suboption of -qipa 172  
 pwr suboption of -qarch 114  
 pwr suboption of -qtune 240  
 pwr2 suboption of -qarch 114  
 pwr2 suboption of -qtune 240  
 pwr2s suboption of -qarch 115  
 pwr2s suboption of -qtune 240  
 pwr3 suboption of -qarch 116  
 pwr3 suboption of -qtune 240  
 pwr4 suboption of -qarch 116  
 pwr5 suboption of -qarch 116  
 pwr5x suboption of -qarch 116  
 pwrx suboption of -qarch 114  
 pwrx suboption of -qtune 240

## Q

QCOUNT @PROCESS directive 209  
 qdirectstorage compiler option 137  
 quiet NaN 164  
 quiet NaN suboption of -qfltrap 154

## R

rand routine, naming restriction for 36  
 random run-time option 48  
 READ statements past end-of-file 252  
 READMExlf file 7  
 REAL data types 122  
 REALSIZE @PROCESS directive 210

rec\_locks suboption of -qsmp 233  
 RECUR @PROCESS directive 212  
 recursion 212, 216  
 register flushing 176  
 REPORT @PROCESS directive 213  
 resetpdf command 199  
 return code  
   from compiler 276  
   from Fortran programs 276  
 rndsngl suboption of -qfloat 153  
 rrm suboption of -qfloat 153, 230  
 rsqrt suboption of -qfloat 153  
 run time  
   exceptions 49  
   options 40  
 run-time  
   libraries 23, 33, 34  
   problems 281  
   SMP libraries 31, 32  
 run-time environment  
   external names in 304  
 running a program 37  
 running the compiler 20  
 runtime suboption of  
   -qsmp=schedule 224

## S

S error severity 275  
 SAA @PROCESS directive 215  
 SAA FORTRAN definition, checking  
 conformance to 4  
 safe suboption of -qipa 172  
 SAVE @PROCESS directive 216  
 schedule suboption of -qsmp 223  
 scratch file directory  
   *See* TMPDIR environment variable  
 scratch\_vars run-time option 11, 48  
 segmentation fault 198  
 setlocale libc routine 40  
 setrteopts service and utility  
   procedure 40  
 severe error 275  
 sh shell 8  
 shared libraries 303  
 shared object files 23  
 SIGN intrinsic, effect of  
   -qxl90=signedzero on 254  
 signal handling 49  
 signedzero suboption of -qxl90 254  
 SIGTRAP signal 49, 154  
 small and large suboptions of -qpic 204  
 small suboption of -qtbtable 238  
 SMP  
   programs, compiling 22  
   smplibraries attribute of configuration  
   file 12  
 smplist suboption of -qreport 213  
 softeof suboption of -qxl77 252  
 SOURCE @PROCESS directive 227  
 source file options 27  
 source files 23  
   allowing suffixes other than .f 12  
   preserving path names for  
   debugging 157  
   specifying options in 27  
 source section in compiler listing 296

- source-code conformance checking 4
- source-level debugging support 5
- space problems 279
- space, increasing amount for data and stack 85
- SPILLSIZE @PROCESS directive 99, 228
- ssuffix attribute of configuration file 12
- stack
  - limit 85, 279
- static linking 36
- static storage, alignment of arrays in 112
- static suboption of -qsmp=schedule 224
- std suboption of -qalias 109
- stdexits suboption of -qipa 173
- storage limits 279
- storage relationship between data objects 305
- storage-associated arrays, performance implications of 109
- STRICT @PROCESS directive 230
- strictieemod @PROCESS directive 231
- strictnmaf suboption of -qfloat 153
- strings, passing to C functions 194
- struct command 302
- subprogram calls to other languages
  - See subprograms in other languages, calling
- suffix, allowing other than .f on source files 12
- suffixes for source files 233
- summary of compiler options 51
- SWAPOMP @PROCESS directive 236
- symbolic debugger support 5
- system problems 279

## T

- target machine, compiling for 114
- temporary arrays, reducing 109
- temporary file directory 10
- temporary files
  - See /tmp directory
- text editors 19
- TextEdit text editor 19
- threads suboption of -qipa 173
- threads, controlling 44
- threshold suboption of -qsmp 224
- throughput for I/O, increasing with data striping 112
- times routine, naming restriction for 36
- TMPDIR environment variable 49, 280
  - compile time 10
- tprof command 238
- Trace/BPT trap 49
- traceback listing 220, 282
- transformation report section in compiler listing 297
- trigger\_constant
  - \$OMP 222
  - IBM\* 135
  - IBMP 222
  - IBMT 239
  - setting values 135
  - SMP\$ 222
- trigraphs 30
- tuning performance
  - See optimization

- typeless constants and character constants 131
- typesmt and notypesmt suboptions of -qport 205

## U

- U error severity 275
- ulimit command 279
- UNDEF @PROCESS directive 243, 263
- underflow suboption of -qflttrap 154
- Unicode data 189
- unit\_vars run-time option 11, 48, 49
- UNIVERSAL setting for locale 189
- unknown suboption of -qipa 173
- unrecoverable error 275
- unresolved references, fixing via -brename option 72
- unrolling DO LOOPS 244
- UNWIND @PROCESS directive 245
- upgrading to the latest version of XL Fortran 15
- use attribute of configuration file 12
- UTF-8 encoding for Unicode data 189
- uwidth run-time option 48

## V

- value relationships between data objects 305
- vi text editor 19

## W

- W error severity 275
- warning error 275
- what command 14, 302
- WRITE statements past end-of-file 252

## X

- XFLAG(OLDTAB) @PROCESS directive 250
- XFLAG(XALIAS) @PROCESS directive (obsolete) 251
- xl\_trbk library procedure 282
- xl\_trce exception handler 220
- XL\_NOCLONEARCH environment variable 49
- xl attribute of configuration file 12
- xl command
  - description 20
  - level of Fortran standard compliance 15, 21
- xl\_r command
  - description 20
  - for compiling SMP programs 22
  - level of Fortran standard compliance 15, 21
- xl\_r7 command
  - description 20
  - for compiling SMP programs 22
  - level of Fortran standard compliance 15, 21
- xl.cfg configuration file 93

- XL77 @PROCESS directive 252
- XL90 @PROCESS directive 254
- xl90 command
  - description 20
  - level of Fortran standard compliance 15, 21
- xl90\_r command
  - description 20
  - for compiling SMP programs 22
  - level of Fortran standard compliance 15, 21
- xl90\_r7 command
  - description 20
  - for compiling SMP programs 22
  - level of Fortran standard compliance 15, 21
- xl95 command
  - description 20
  - level of Fortran standard compliance 15
- xl95\_r command
  - description 20
  - for compiling SMP programs 22
  - level of Fortran standard compliance 15, 21
- xl95\_r7 command
  - description 20
  - for compiling SMP programs 22
  - level of Fortran standard compliance 15, 21
- xlentry file 302
- xlfopt attribute of configuration file 12
- XLFRTEOPTS environment variable 40
- XLFSCRATCH\_unit environment variable 11, 48
- XLFUNIT\_unit environment variable 11, 48
- XLINES @PROCESS 256
- XOR 252
- XREF @PROCESS directive 258
- xrf\_messages run-time option 48

## Z

- z suboption of -y 269
- zero suboption of -qieee 162
- zerodivide suboption of -qflttrap 154
- zeros (leading), in output 252
- ZEROSIZE @PROCESS directive 259





Program Number: 5724-M13

SC09-8007-00

