IBM XL Fortran Enterprise Edition V10.1 for AIX

Getting Started with XL Fortran

SC09-8009-00

IBM XL Fortran Enterprise Edition V10.1 for AIX

Getting Started with XL Fortran

SC09-8009-00

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 37.

First Edition (September 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.

Contents

About this document

Getting Started with XL Fortran provides a general overview of the XL Fortran compiler, its more significant features, and how those features can help you improve your software development productivity.

For the benefit of current XL Fortran users upgrading to this release, *Getting Started with XL Fortran* also includes a summary of features that are new or improved for V10.1.

Getting Started with XL Fortran is intended only to help familiarize you with the compiler. For detailed information on using the XL Fortran compiler, you will want to refer to other books in the XL Fortran Enterprise Edition V10.1 for AIX library of books, described in "IBM XL Fortran publications" on page viii.

Who should read this document

Getting Started with XL Fortran is intended for anyone who plans to work with IBM® XL Fortran Enterprise Edition V10.1 for AIX, who is familiar with the AIX operating system, and who has some previous Fortran programming experience.

How to use this document

If you are new to XL Fortran , you should view Chapter 1, "Overview of XL Fortran features," on page 1 to familiarize yourself with the key features of XL Fortran and how to begin using it to develop your applications.

If you are already an experienced XL Fortran user and are now upgrading to the latest release of XL Fortran , you may want to go directly to Chapter 2, "What's new for V10.1," on page 7 to review that latest changes and feature enhancements to the compiler.

The remaining sections of this guide provide a brief overview of basic program development tasks with XL Fortran.

How this document is organized

This guide includes these topics:

- Chapter 1, "Overview of XL Fortran features," on page 1 outlines the the key features of the XL Fortran compiler
- Chapter 2, "What's new for V10.1," on page 7 describes new and updated features offered by the latest version of XL Fortran.
- Chapter 3, "Setting up and customizing XL Fortran," on page 21 provides brief overview information on the steps involved in setting up and customizing XL Fortran, together with pointers on where you can find more detailed information.
- Chapter 4, "Editing, compiling, and linking programs with XL Fortran," on page 23 discusses the basic steps involved in creating and compiling your applications with XL Fortran.
- Chapter 5, "Running XL Fortran programs," on page 31 describes how to run your compiled applications, including setting of run time options.

- Chapter 6, "XL Fortran compiler diagnostic aids," on page 33 offers guidance on how to use XL Fortran compiler diagnostic aids to identify and correct compilation problems with your applications.

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$:$OMP:IBMP:IBMT** will be on.
italics	Parameters or variables whose actual names or values are to be supplied by the user. Italics are also used to introduce new terms.	The maximum length of the *trigger_constant* in fixed source form is 4 for directives that are continued on one or more lines.
UPPERCASE	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.
`monospace`	Programming keywords and library functions, compiler built-in functions, 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

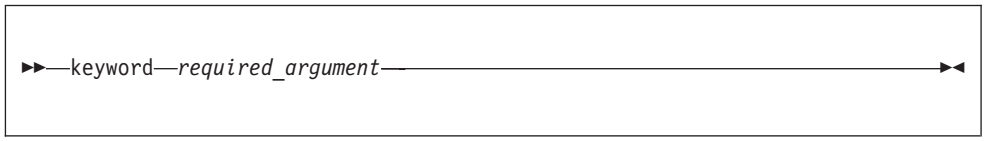
Throughout this document, diagrams illustrate XL Fortran syntax. This section will help you to interpret and use those diagrams.

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

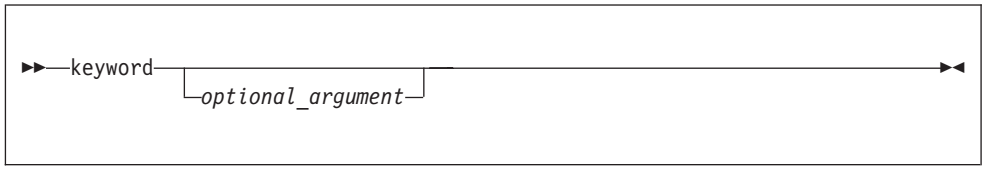
You must enter punctuation marks, parentheses, arithmetic operators, and other special characters as part of the syntax.

- Read syntax diagrams from left to right and from top to bottom, following the path of the line:
  - The ►►— symbol indicates the beginning of a statement.
  - The —► symbol indicates that the statement syntax continues on the next line.
  - The ►— symbol indicates that a statement continues from the previous line.
  - The —►◄ 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 and Fortran 95 extensions 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):



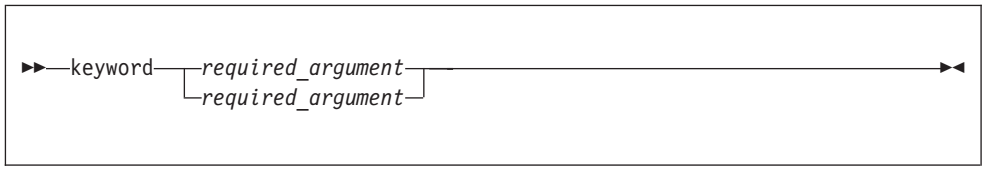
- Optional items are shown below the main path:



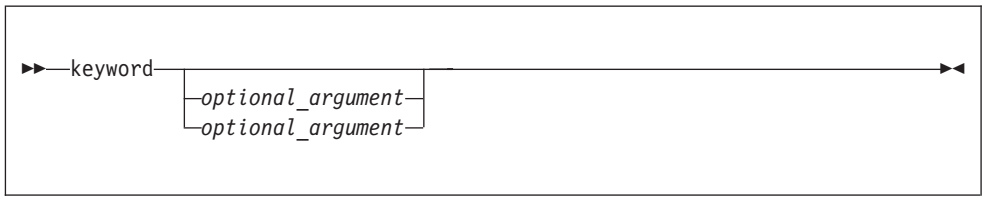
Note: Optional items (not in syntax diagrams) are enclosed by square brackets ([and]). For example, `[UNIT=]u`

- If you can choose from two or more items, they are shown vertically, in a stack.

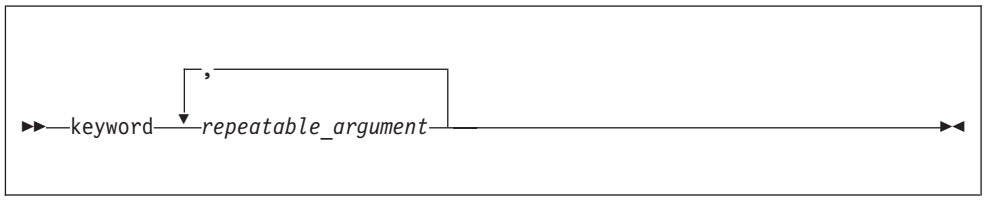
  If you *must* choose one of the items, one item of the stack is shown on the main path:



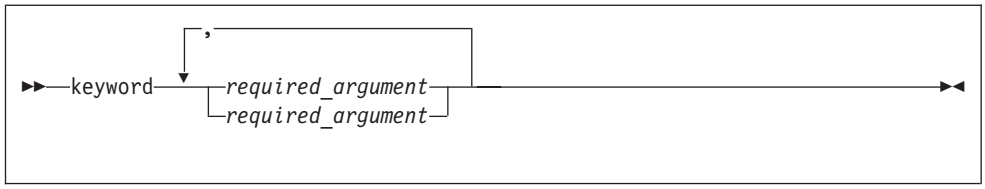
  If choosing one of the items is optional, the entire stack is shown below the main path:



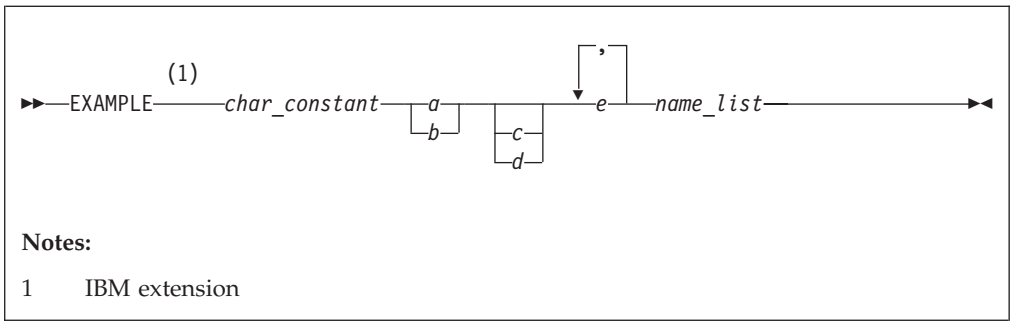
- An arrow returning to the left above the main line (a repeat arrow) indicates that you can repeat an item, and the separator character if it is other than a blank:



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



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*.)

Examples

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

Related information

IBM XL Fortran publications

XL Fortran provides product documentation in the following formats:

- Readme files

  Readme files contain late-breaking information, including changes and corrections to the product documentation. Readme files are located by default in the /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 *IBM XL Fortran Enterprise Edition V10.1 for AIX Installation Guide*. The information center is also viewable on the Web at:

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, and 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
IBM XL Fortran Enterprise Edition V10.1 for AIX Installation Guide, GC09-8008-00	install.pdf	Contains information for installing XL Fortran and configuring your environment for basic compilation and program execution.
IBM XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference, GC09-8007-00	cr.pdf	Contains information about the various compiler options and environment variables.
IBM XL Fortran Enterprise Edition V10.1 for AIX Language Reference, GC09-8006-00	lr.pdf	Contains information about the Fortran programming language as supported by IBM, including language extensions for portability and conformance to non-proprietary standards, compiler directives and intrinsic procedures.
IBM XL Fortran Enterprise Edition V10.1 for AIX Optimization and Programming Guide, GC09-8010-00	opg.pdf	Contains information on advanced programming topics, such as application porting, interlanguage calls, floating-point operations, input/output, application optimization and parallelization, and the XL Fortran high-performance libraries.

These PDF files are viewable and printable from Adobe Reader. If you do not have the Adobe Reader installed, you can download it from:

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

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. Overview of XL Fortran features

XL Fortran Enterprise Edition V10.1 for AIX can be used for large, complex, computationally intensive programs, including interlanguage calls with C and C++ programs. This section discusses the features of the XL Fortran compiler at a high level. It is intended for people who are evaluating XL Fortran and for new users who want to find out more about the product.

Commonality with other XL compilers

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

The XL compilers are part of a larger family of IBM C, C++, and Fortran compilers that are derived from a common code base that shares compiler function and optimization technologies among a variety of platforms and programming languages, such as AIX, Linux distributions, OS/390, OS/400, z/OS, and z/VM operating systems. The common code base, along with compliance to international programming language standards, helps ensure consistent compiler performance and ease of program portability across multiple operating systems and hardware platforms.

The XL compilers are available for use on AIX and select Linux distributions.

Documentation, online help, and technical support

This guide provides an overview of XL Fortran and its features. You can also find more extensive product documentation in the following formats:

- Readme files.
- Installable man pages.
- A searchable, HTML-based help system.
- Portable Document Format (PDF) documents.
- Online technical support over the Web.

For more information about product documentation and technical support provided with XL Fortran, see:

- "IBM XL Fortran publications" on page viii
- "Additional documentation" on page ix
- "Technical support" on page ix

Hardware and operating system support

XL Fortran Enterprise Edition V10.1 for AIX supports AIX 5L for POWER V5.1, V5.2, and V5.3. See the README file and Installing XL Fortran Enterprise Edition in the *XL Fortran Enterprise Edition V10.1 for AIX Installation Guide* for a complete list of requirements.

The compiler, its libraries, and its generated object programs will run on all RS/6000® or pSeries® systems with the required software and disk space.

All supported processors other than POWER and POWER2 are considered part of the PowerPC family. Any reference to PowerPC includes all chips except POWER and POWER2

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.

Highly configurable compiler

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

Compiler invocation commands

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

The compiler also provides corresponding "**_r**" versions of most invocation commands, for example, **xlf_r** and **xlf_r7**. These "**_r**" invocations instruct the compiler to link and bind object files to thread-safe components and libraries, and produce threadsafe object code for compiler-created data and procedures.

For more information about XL Fortran compiler invocation commands, see "Compiling with XL Fortran" on page 24 in this book or Compiling XL Fortran programs in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference*.

Compiler options

You can control the actions of the compiler through a large set of provided compiler options. Different categories of options help you to debug your applications, optimize and tune application performance, select language levels and extensions for compatibility with programs from other platforms, and do many other common tasks that would otherwise require changing the source code.

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

For more information about XL Fortran compiler options, see Compiler-options reference in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference*.

Custom compiler configuration files

The installation process creates a default plain text compiler configuration file at /etc/xlf.cfg. This configuration file contains several stanzas that define compiler option default settings.

Your compilation needs may frequently call for specifying compiler option settings other than the defaults settings provided by XL Fortran. If so, you can create your own custom configuration files containing your own frequently-used compiler option settings, and call those configuration files when you compile your applications.

See Customizing the configuration file in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference* for more information on creating and using custom configuration files.

Language standards compliance

The compiler supports the following programming language specifications for Fortran:

- ISO/IEC 1539-1:1991(E) and ANSI X3.198-1992 (referred to as Fortran 90 or F90)
- ISO/IEC 1539-1:1997 (referred to as Fortran 95 or F95)
- Extensions to the Fortran 95 standard:
  - Industry extensions that are found in Fortran products from various compiler vendors
  - Extensions specified in SAA Fortran
- Partial support of the Fortran 2003 standard

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

- OpenMP extensions to support parallelized programming.
- Language extensions to support VMX vector programming.

See Language standards in the *XL Fortran Enterprise Edition V10.1 for AIX Language Reference* for more information about Fortran language specifications and extensions.

Source-code migration and conformance checking

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

To protect your investments in FORTRAN 77 object code, you can link Fortran 90 and Fortran 95 programs with existing FORTRAN 77 object modules and libraries.

See **-qlanglvl** in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference* for more information.

Program optimization

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

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

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

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

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

If possible, you should test and debug your code without optimization before attempting to optimize it.

For more information about optimization techniques, see Optimizing XL compiler applications in the *XL Fortran Enterprise Edition V10.1 for AIX Optimization and Programming Guide*.

For a summary of optimization-related compiler options, see Options for performance optimization in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference*.

64-bit object capability

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

To support larger executables that can be fit within a 64-bit address space, a separate, 64-bit object form is used 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 explicitly attempts to load a 32-bit module
- A 32-bit executable that explicitly attempts to load a 64-bit module
- Attempts to run 64-bit applications on 32-bit platforms

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

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

For more information, see Using XL Fortran in a 64-Bit Environment in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference*.

Shared memory parallelization

XL Fortran Enterprise Edition V10.1 for AIX supports application development for multiprocessor system architectures. You can use any of the following methods to develop your parallelized applications with XL Fortran:

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

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

For more information, see Parallel programming with XL Fortran in the *XL Fortran Enterprise Edition V10.1 for AIX Optimization and Programming Guide*.

OpenMP directives

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

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

OpenMP directives address three important issues of parallelizing an application:

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

XL Fortran supports the OpenMP API Version 2.5 specification. For more information, see www.openmp.org.

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 "XL Fortran compiler listings" on page 33.

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

Symbolic debugger support

You can use **dbx** or any other symbolic debugger that supports the AIX XCOFF executable format when debugging your programs.

Chapter 2. What's new for V10.1

The new features and enhancements in XL Fortran Enterprise Edition V10.1 for AIX fall into four categories:

Performance and optimization

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

Architecture and processor-specific code tuning

The **-qarch** compiler option controls the particular instructions that are generated for the specified machine architecture. The **-qtune** compiler option adjusts the instructions, scheduling, and other optimizations to enhance performance on the specified hardware. These options work together to generate application code that gives the best performance for the specified architecture.

XL Fortran V10.1 augments the list of suboptions available to the **-qarch** compiler option to support newly-available POWER5+ processors and processors that support the VMX instruction set. The following new **-qarch** options are available:

- -qarch=pwr5x
- -qarch=ppc64v

High performance libraries

XL Fortran includes highly-tuned mathematical functions that can greatly improve the performance of mathematically-intensive applications. These functions are provided through the following high-performance libraries:

Mathematical Acceleration Subsystem (MASS)

MASS libraries provide high-performance scalar and vector functions to perform common mathematical computations. The MASS libraries included with XL Fortran Enterprise Edition V10.1 for AIX introduce new scalar and vector functions, and new support for the POWER5 processor architecture.

For more information about using the MASS libraries, see Using the Mathematical Acceleration Subsystem in the *XL Fortran Enterprise Edition V10.1 for AIX Optimization and Programming Guide*.

Basic Linear Algebra Subprograms (BLAS)

XL Fortran Enterprise Edition V10.1 for AIX introduces the BLAS set of high-performance algebraic functions. You can use these functions to:

- Compute the matrix-vector product for a general matrix or its transpose.
- Perform combined matrix multiplication and addition for general matrices or their transposes.

For more information about using the BLAS functions, see Using the Basic Linear Algebra Subprograms in the *XL Fortran Enterprise Edition V10.1 for AIX Optimization and Programming Guide*.

VMX support

XL Fortran now provides a VECTOR data type and vector multimedia extension (VMX) intrinsic functions to support direct AltiVec programming.

Objects compiled with VECTOR data types and related operations can run on systems with processor architectures and operating systems (AIX 5L Version 5.3 with the 5300-03 Recommended Maintenance package or higher) that support the single instruction, multiple data (SIMD) instruction set. The SIMD instruction set (also known as vector multimedia extension or VMX instructions) enables higher utilization of microprocessor hardware and supports performing calculations in parallel. The compiler provides the ability to automatically enable SIMD vectorization at higher levels of optimization.

This release of XL Fortran introduces several new option and suboption combinations to enable and exploit VMX instructions.

Table 3. VMX-Related Compiler Options and Directives

Option/directive	Description
-qenablevmx | **-qnoenablevmx**	Setting **-qenablevmx** enables compiler generation of VMX instructions. It also enables Fortran language support for the VECTOR data type and the VMX intrinsic functions. Setting **-qnoenablevmx** disables compiler generation of VMX instructions. This is the compiler default setting. **Note:** You can set **-qenablevmx** (and other options described in this section that cause the compiler to generate VMX instructions) if you are compiling your application on a system that does not support VMX instructions, but are targeting your compiled objects for later use on a system that does support VMX instructions.
-qhot=simd | **-qhot=nosimd**	When **-qhot=simd** is in effect, the compiler will try to improve application performance by converting certain loop operations on successive elements in an array into calls to the faster, more efficient VMX instructions.. This option has effect only when the target architecture supports VMX instructions and **-qenablevmx** is set. When **-qhot=nosimd** is in effect, the compiler performs optimizations on loops and arrays, but does not replace code with calls to VMX instructions.
-qvecnvol | **-qnovecnvol**	**-qvecnvol** instructs the compiler to generate objects that use both volatile and non-volatile vector registers, providing potential performance benefits on systems that support VMX instructions. **-qnovecnvol** instructs the compiler to generate objects that use only volatile vector registers. Volatile vector registers do not preserve their values across function calls or context save/jump/switch system library functions. Setting this option will make your vector applications safe where there is risk of interaction with objects built with AIX libraries prior to AIX 5.3 with 5300-03, but may also result in reduced application performance. This is the compiler default setting.

Other performance-related compiler options and directives

The entries in the following table describes new or changed compiler options and directives not already mentioned in the sections above.

Information presented here is just a brief overview. For more information about these compiler options, refer to Options for performance optimization in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference*.

Table 4. Other Performance-Related Compiler Options and Directives

Option/directive	Description
-qhot	**-qhot** adds the following new suboptions: **-qhot=level=0** The compiler performs a subset of high-order transformations. • -qhot=novector • -qhot=nosimd • -qhot=noarraypad This setting is the default when **-O3** optimization is in effect. **-qhot=level=1** The compiler performs the complete range of high-order transformations. • -qhot=vector • -qhot=simd • -qhot=arraypad This setting is the default when **-O4** or **-O5** optimization is in effect. **-qhot=simd** Described above in "VMX support" on page 8.

Table 4. Other Performance-Related Compiler Options and Directives (continued)

Option/directive	Description
-qipa	**-qipa** adds the following new suboptions: **-qipa=clonearch=***arch*{,*arch*} Specifies one or more processor architectures for which multiple versions of the same instruction set are produced. XL Fortran lets you specify multiple specific processor architectures for which instruction sets will be generated. At run time, the application will detect the specific architecture of the operating environment and select the instruction set specialized for that architecture. **-qipa=cloneproc=***name*{,*name*} Specifies the names of one or more functions to clone for the processor architectures specified by the **clonearch** suboption. **-qipa=malloc16** This new option has effect only at link time. It asserts to the compiler that dynamic memory allocation routines such as malloc, calloc, realloc, and new will return addresses aligned on 16-byte boundaries, and instructs the compiler to optimize generated code according to that assertion. This option is set by default when compiling in 64-bit mode, but can be overridden with **-qipa=nomalloc16**. **Notes:** 1. You must specify **-qipa=nomalloc16** only if you can ensure that executables created with this option will be run in an environment where dynamic memory allocations can return addresses aligned on 16-byte boundaries. 2. If you are using **-qhot=simd**, you should also consider specifying **-qipa=malloc16** to expose additional VMX optimization opportunities.
-O	Specifying the **-O3** compiler option now instructs the compiler to also assume the **-qhot=level=0** compiler option setting. Specifying the **-O4** or **-O5** compiler option now instructs the compiler to also assume the **-qhot=level=1** compiler option setting.

Table 4. Other Performance-Related Compiler Options and Directives (continued)

Option/directive	Description
-qsmallstack	The **-qsmallstack** compiler option adds the following new suboptions to control dynamic length variable allocation transformations: **-qsmallstack=dynlenonheap** When this suboption is specified, certain automatically-sized objects are allocated from the heap. This suboption affects automatic objects that have nonconstant character lengths or a nonconstant array bound (DYNamic LENgth ON HEAP). Specifying this suboption turns on both dynlenonheap and general smallstack transformations. **-qsmallstack=nodynlenonheap** This is the default. If this suboption is not specified, those objects are allocated on the stack. This suboption affects automatic objects that have nonconstant character lengths or a nonconstant array bound (DYNamic LENgth ON HEAP).
-qstacktemp	The **-qstacktemp** compiler option is new, and gives you the ability to control where certain compiler temporaries are stored. Available suboptions are: **-qstacktemp=0** This is the default. Certain compiler temporaries are allocated to the heap instead of the stack at compiler's discretion, depending on the size of the compiler temporaries and the target operating system environment. **-qstacktemp= -1** Certain compiler temporaries are always allocated on the stack, providing best performance but also using the most amount of stack space. **-qstacktemp=***num_bytes* Certain compiler temporaries less than *num_bytes* in size are allocated to the stack. Compiler temporaries greater than or equal to *num_bytes* are allocated to the heap. Programs that use large arrays may to use this option if they are running out of stack space at run time. SMP or OpenMP applications that are constrained by stack space may also find this option useful to move some compiler temporaries onto the heap from the stack.

Intrinsic procedures new for this release

The following table lists intrinsic procedures that are new for this release. For more information on intrinsic procedures provided by XL Fortran, see Intrinsic procedures in the *XL Fortran Enterprise Edition V10.1 for AIX Language Reference*.

Table 5. Intrinsic procedures for XL Fortran

Function	Description
`FRIM(val);`	Takes an input *val* of REAL *8 format, rounds *val* down to the next lower integral value, and returns the result in REAL *8 format. Valid only for POWER5+ processors.
`FRIMS(val);`	Takes an input *val* in REAL *4 format, rounds *val* down to the next lower integral value, and returns the result in REAL *4 format. Valid only for POWER5+ processors.

Table 5. Intrinsic procedures for XL Fortran (continued)

Function	Description
`FRIN(val);`	Takes an input *val* in REAL *8 format, rounds *val* to the nearest integral value, and returns the result in REAL *8 format. Valid only for POWER5+ processors.
`FRINS(val);`	Takes an input *val* in REAL *4 format, rounds *val* to the nearest integral value, and returns the result in REAL *4 format. Valid only for POWER5+ processors.
`FRIP(val);`	Takes an input *val* in REAL *8 format, rounds *val* up to the next higher integral value, and returns the result in REAL *8 format. Valid only for POWER5+ processors.
`FRIPS(val);`	Takes an input *val* in REAL *4 format, rounds *val* up to the next higher integral value, and returns the result in REAL *4 format. Valid only for POWER5+ processors.
`FRIZ(val);`	Takes an input *val* in REAL *8 format, rounds *val* to the next integral value closest to zero, and returns the result in REAL *8 format. Valid only for POWER5+ processors.
`FRIZS(val);`	Takes an input *val* in REAL *4 format, rounds *val* to the next integral value closest to zero, and returns the result in REAL *4 format. Valid only for POWER5+ processors.

Support for language enhancements and APIs

API and language enhancements can offer you additional ease of use and flexibility when developing your applications, as well as making it easier for you to develop code that more fully exploits the capabilities of your hardware platform.

XL Fortran language enhancements

XL Fortran V10.1 implements new features compliant to the Fortran 2003 standard. These features, supported when **-qlanglvl=2003std** or **-qlanglvl=2003pure** is in effect, include:

ENUM statement
You can specify an ENUM statement to define and group a set of named integer constants called enumerators. The storage size of the enumerators can be set by the new **-qenum** compiler option.

READ statement - new BLANK and PAD specifiers
The BLANK specifier controls the default interpretation of blanks when you are using a format specification. The setting of the PAD specifier determines if input records are padded with blanks.

WRITE statement - new DELIM specifier
The DELIM specifier specifies what delimiter, if any, is used to delimit character constants written with list-directed or namelist formatting.

Relaxed rules for specification expression
XL Fortran now allows recursive functions to be specification functions.

Procedure pointers
This release adds support for procedure pointers. A procedure pointer is a PROCEDURE entity that has the EXTERNAL and POINTER attribute. It may be a pointer associated with an external procedure, a module procedure, an intrinsic procedure, or a dummy procedure that is not a procedure pointer.

See *IBM XL Fortran Enterprise Edition V10.1 for AIX Language Reference* for more information.

OpenMP API V2.5 support for C, C++, and Fortran

XL Fortran now supports the OpenMP API V2.5 standard. This latest level of the OpenMP specification combines the previous C/C++ and Fortran OpenMP specifications into one single specification for both C/C++ and Fortran, and resolves previous inconsistencies between them.

The OpenMP Application Program Interface (API) is a portable, scalable programming model that provides a standard interface for developing user-directed shared-memory parallelization in C, C++, and Fortran applications. The specification is defined by the OpenMP organization, a group of computer hardware and software vendors, including IBM.

You can find more information about OpenMP specifications at:

www.openmp.org

Ease of use

XL Fortran includes the following new features to help you more easily use the compiler for your application development.

Newly-supported filename extensions

XL Fortran Enterprise Edition V10.1 for AIX adds support for the .f77, .f90, and .f95 filename extensions.

You can use generic XL Fortran compiler invocations, such as **xlf** and **xlf_r**, to compile program source files with these filename extensions. The compiler will recognize the filename extensions and apply the appropriate language standard defaults as if you were compiling using the **f77**, **xlf90**, or **xlf95** compiler invocations and their associated **_r** counterparts.

Support for IBM Tivoli License Manager

IBM Tivoli License Manager (ITLM) is a Web-based solution that can help you manage software usage metering and license allocation services on supported systems. In general, ITLM recognizes and monitors the products that are installed and in use on your system.

IBM XL Fortran Enterprise Edition V10.1 for AIX is ITLM-enabled for inventory support only, which means that ITLM is able to detect product installation of XL Fortran, but not its usage.

Note: ITLM is not a part of the XL Fortran compiler offering, and must be purchased and installed separately.

Once installed and activated, ITLM scans your system for product inventory signatures that indicate whether a given product is installed on your system. ITLM also identifies that product's version, release, and modification levels. Signature files for XL Fortran are installed to the following directory:

Default installations
`/usr/lpp/xlf`

Non-default installations
compiler/usr/lpp/xlf where *compiler* is the target directory for installation specified by the **-b** installation option.

For more information about IBM Tivoli License Manager Web, see:

www.ibm.com/software/tivoli/products/license-mgr

New compiler options

Compiler options can be specified on the command line or through directives embedded in your application source files. The following table describes new compiler options or suboptions not already described elsewhere in this section.

New command line options

The following table summarizes command line options new to XL Fortran. You can find detailed syntax and usage information for all compiler options in Compiler options reference in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference*.

Option	Description and remarks
-qenum	The **-qenum** compiler option specifies the amount of storage used by enumerators defined with the ENUM statement.
-qlanglvl	This release adds the **2003std** and **2003pure** suboptions to the **-qlanglvl** compiler option. **2003std** 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. **2003pure** The same as 2003std except that it also reports errors for any obsolescent Fortran 2003 features used.
-qlibansi	This option is now recognized by the entire compiler, and not just by the IPA optimizer. It instructs the compiler to assume that all functions with the name of an ANSI C defined library function are in fact the library functions.
-qlinedebug	This new compiler option enables minimal generation (line number and source file name) of information for use by a debugger. This compiler option can be specified on the command line or in your program source code as a @PROCESS statement.
-qlist	The **-qlist** compiler option adds new **offset** and **nooffset** suboptions. Specifying **-qlist=offset** instructs the compiler to show object listing offsets from the start of a function rather than from the start of code generation.
-qport	This release adds **clogicals** and **noclogicals** as new suboptions to the **-qport** compiler option. Specifying **-qport=clogicals** together with **-qintlog** instructs the compiler to treat all non-zero integers used in logical expressions as TRUE. This option is useful when porting applications from other Fortran compilers that expect this behavior.

New directives

The following table summarizes directive options new to XL Fortran. You can find detailed syntax and usage information in Directives in the *XL Fortran Enterprise Edition V10.1 for AIX Language Reference*.

Directive	Description and remarks
NOSIMD	The **NOSIMD** directive prohibits the compiler from automatically generating vector multimedia extension (VMX) instructions in the loop immediately following the directive, or in the FORALL construct.
NOVECTOR	The **NOVECTOR** directive prohibits the compiler from auto-vectorizing the loop immediately following the directive, or in the FORALL construct. Auto-vectorization refers to converting certain operations performed in a loop and on successive array elements, into a call to a routine that computes several results simultaneously.

If you have just upgraded to XL Fortran Version 10.1

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 10.1

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 **xlf90**, **xlf90_r**, and **xlf90_r7** commands provide Fortran 90 conformance, and the **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 **libxlf90.a** library located in **/usr/lib** provides Fortran 90 and Fortran 95 support. 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 6. Potential Problems Migrating Programs to XL Fortran V10. 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 **-qextern** option, or insert **EXTERNAL** statements to avoid the ambiguity. Consider switching to the Fortran 90 or Fortran 95 procedure if it does what you want.	
The **.XOR.** intrinsic is not recognized.	Use the option **-qxlf77=intxor**.	✔
Zero-sized objects are not allowed by the compiler.	Use the **xlf90** or **xlf95** command, or use the **-qzerosize** option with the **xlf** or **f77** command.	
Performance / optimization problems		
Existing programs or programs linked with older XL Fortran object files run more slowly or do not show expected performance improvements on new hardware.	Recompile everything.	
Programs compiled with **-O3** or **-qhot** optimization behave differently from those unoptimized (different results, exceptions, or compilation messages).	Try adding the **-qstrict** option.	

Table 6. Potential Problems Migrating Programs to XL Fortran V10 (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?
The option combination **-O** and **-1** cannot be abbreviated **-O1**, to avoid misunderstandings. (There are **-O2**, **-O3**, **-O4**, and **-O5** optimization levels, but there is no **-O1**.)	Specify **-O** and **-1** as separate options.	
Programs that use integer **POINTER**s produce incorrect results when optimized.	Specify the option **-qalias=intptr** with the **xlf90** or **xlf95** command, or use the **xlf** command.	✔

Table 6. Potential Problems Migrating Programs to XL Fortran V10 (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?
Runtime problems		
Programs that read to the end of the file and then try to append records without first executing a **BACKSPACE** statement do not work correctly. The write requests generate error messages.	To compile existing programs, specify the option **-qxlf77=softeof** with the **xlf90** or **xlf95** command, or use the **xlf** command. For new programs, add the **BACKSPACE** statement before writing past the endfile record.	✔
Uninitialized variables are not necessarily set to zero, and programs that ran before may exceed the user stack limit. The reason is that the default storage class is now **AUTOMATIC**, rather than **STATIC** (an implementation choice allowed by the language).	Ensure that you explicitly initialize your variables, use the **-qsave** option with the **xlf90** or **xlf95** command, or add **SAVE** statements where needed in the source.	✔
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.	✔
Numeric results and I/O output are not always exactly identical with XL Fortran Version 2. Certain implementation details of I/O, such as spacing in list-directed output and the meanings of some **IOSTAT** values, have changed since XL Fortran Version 2. (This entry is similar to the previous one except that these differences 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 **-qxlf77** compiler option or **XLFRTEOPTS** runtime options, there is no way to get the old behavior back.	

Table 6. Potential Problems Migrating Programs to XL Fortran V10 (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?
SIGN(A,B) now returns **-|A|** when **B=-0.0**. Prior to XL Fortran Version 7.1, it returned **|A|**.	This behavior conforms with the Fortran 95 standard and is consistent with the IEEE standard for binary floating-point arithmetic. It occurs because the **-qxlf90=signedzero** 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 **-qxlf90=signedzero** option is turned on. Turn it off, or specify a command that does not use this option by default.	✓

Chapter 3. Setting up and customizing XL Fortran

This section provides brief overview information about setting up and customizing XL Fortran, together with pointers to other documentation that describes specific set-up and customization topics in greater detail.

Environment variables and XL Fortran

XL Fortran uses a number of environment variables to control various aspects of compiler operation. Environment variables fall into two basic categories:

- Environment variables defining the basic working environment for the compiler.
- Environment variables defining run time compiler option defaults.

Setting the compiler working environment

These environment variables define the basic working environment for the compiler, including specifying your choice of national language or defining the location of libraries or temporary files. For complete information, refer to Correct settings for environment variables in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference*.

LANG
Specifies the default national language *locale* used to display diagnostic messages and compiler listings.

LIBPATH
Specifies the location of libraries used at run time.

MANPATH
Specifies the search path for system, compiler, and third-party man pages.

NLSPATH
Specifies one or more directory locations where message catalogs can be found.

PDFDIR
Specifies the directory location where profile-directed feedback information is stored when you compile with the **-qpdf** option.

TMPDIR
Specifies the directory location where the compiler will store temporary files created during program compilation.

XLFSCRATCH_unit
Used to give a specific name to a scratch file.

XLFUNIT_unit
Used to give a specific name to an implicitly connected file or a file opened with no **FILE=** specifier.

Setting the default runtime options

These environment variables define runtime compiler option defaults to be used by the compiler, unless explicitly overridden by compiler option settings specified on the command line or in directives located in your program source.

OBJECT_MODE
If set, this environment variable sets the default object mode to be either

32-bit or 64-bit, or both. See Default bit mode in the XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference for more information.

XLFRTEOPTS
The **XLFRTEOPTS** environment variable allows you to specify options that affect I/O, EOF error-handling, and the specification of random-number generators. See Setting run time options in the XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference for more information.

XLSMPOPTS
The **XLSMPOPTS** environment variable allows you to specify run time options that affect SMP execution. See Setting OMP and SMP runtime options in theXL Fortran Enterprise Edition V10.1 for AIX Optimization and Programming Guide for more information.

OMP_DYNAMIC, OMP_NESTED, OMP_NUM_THREADS, OMP_SCHEDULE

These environment variables, are part of the OpenMP standard. They let you specify how the application will execute sections of parallel code. See Setting OMP and SMP runtime options in theXL Fortran Enterprise Edition V10.1 for AIX Optimization and Programming Guide for more information.

Customizing the configuration file

The configuration file is a plain text file that specifies default settings for compiler options and invocations. XL Fortran provides a default configuration at file /etc/xlf.cfg during compiler installation.

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.

As an alternative, you can create additional custom configuration files to meet special compilation requirements demanded by specific applications or groups of applications.

See Customizing the configuration file in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference* for more information on creating and using custom configuration files.

Determining what level of XL Fortran is installed

You may not be sure which level of XL Fortran is installed on a particular machine. You will need to know this information if contacting software support.

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

```
xlf -qversion
```

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

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

Prerequisite Information:

1. Before you can use the compiler, you must first ensure that all AIX settings (for example, certain environment variables and storage limits) are correctly configured. For more information see "Environment variables and XL Fortran" on page 21.
2. To learn more about writing Fortran programs, refer to the *XL Fortran Enterprise Edition V10.1 for AIX Language Reference*.

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 **.s** files and for **.S** files after they are preprocessed)
4. The linker **ld**

To see the compiler step through these phases, specify the **-qphsinfo** and **-v** compiler options when you compile your application.

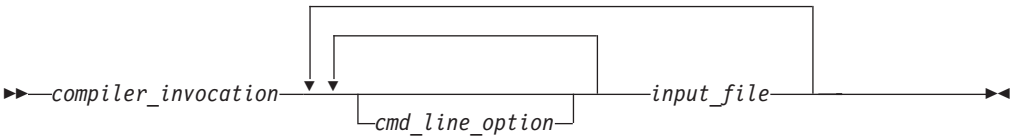
Editing Fortran source files

To create Fortran source programs, you can use any of the available text editors, such as **vi** or **emacs**. Source programs must use a recognized filename suffix unless the configuration file defines additional non-standard filename suffixes. See "XL Fortran input files" on page 26 for a list of filename suffixes recognized by XL Fortran.

For a Fortran source program to be a valid program, it must conform to the language definitions specified in the *XL Fortran Enterprise Edition V10.1 for AIX Language Reference*.

Compiling with XL Fortran

To compile a source program, use one of the compiler invocation commands with the syntax shown below:



The compiler invocation command performs all necessary steps to compile Fortran source files, assemble any **.s** and **.S** files, and link the object files and libraries into an executable program.

When working with source files whose filename extensions indicates a specific level of Fortran, such as .f95, .f90, or f77, compiling with the xlf or threadsafe counterpart invocations will cause the compiler to automatically select the appropriate language-level defaults. The other base compiler invocation commands exist primarily to provide explicit compilation support for different levels and extensions of the Fortran language.

In addition to the base compiler invocation commands, XL Fortran also provides specialized variants of many base compiler invocations. A variation on a base compiler invocation is named by attaching a suffix to the name of that invocation command. Suffix meanings for invocation variants are:

_r Threadsafe invocation variant that supports POSIX Pthread APIs for multithreaded applications, including applications compiled with **-qsmp** or with source code containing IBM SMP or OpenMP program parallelization directives.

_r7 Threadsafe invocation variant that supports Draft 7 POSIX Pthread APIs for multithreaded applications, including applications compiled with **-qsmp** or containing IBM SMP or OpenMP program parallelization directives.

Table 7. XL Fortran compiler invocation commands

Base Invocation	Available Invocation Variants	Description
xlf f77 fort77	xlf_r, xlf_r7	Invokes the compiler so that source files are compiled as FORTRAN 77 source code.
xlf95 f95	xlf95_r, xlf95_r7	Invokes the compiler so that source files are compiled as Fortran 95 source code.
xlf90 f90	xlf90_r, xlf90_r7	Invokes the compiler so that source files are compiled as Fortran 90 source code.

Compiling Fortran 77 programs

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

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

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

Compiling Fortran 95 or Fortran 90 programs

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

f95, xlf95 Fortran 95

f90, xlf90 Fortran 90

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

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

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

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

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

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

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

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

Compiling parallelized XL Fortran applications

XL Fortran provides threadsafe compilation invocations that you can use when compiling parallelized applications for use in multiprocessor environments.

- xlf_r, xlf_r7
- xlf95_r, xlf95_r7
- xlf90_r, xlf90_r7

These invocations are similar to their corresponding base compiler invocations, except that they link and bind compiled objects to threadsafe components and libraries.

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

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. For more information on threadsafe **_r** and **_r7** compiler invocation variants, refer to Compiling XL Fortran programs in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference*.

XL Fortran input files

The input files to the compiler are:

Source files (.f .F ,f77 .F77 .f95 .f90 .F90 .F95 suffixes)
The compiler considers files with these suffixes as being Fortran 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, F77, F95, or F90 are passed through the C preprocessor (**cpp**) before being compiled.

Include files also contain source and often have suffixes different from those ordinarily used for Fortran source files.

Object files (.o suffix)
After the compiler compiles the source files, it uses the **ld** command to link 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. The compiler can then produce a single .o object file or a single executable output file from these object files.

Assembler source files (.s and .S suffixes)
The compiler sends assembler source files to the assembler (**as**). The assembler sends object files to the linker at link time.

Note: Assembler source files with a .S filename suffix are first preprocessed by the compiler, then sent to the assembler.

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

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.

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.

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.

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*.

XL Fortran output files

The output files that Fortran produces are:

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

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 program source input file, and the assembler produces an object file for each specified assembler input file. By default, the output object files are saved to the current directory using the same file name prefixes as their corresponding source input files.

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 input source file. By default, the output assembler source files are saved to the current directory using the same file name prefixes as their corresponding source input files.

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.

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. Compiling modules will also produce .o files that are needed when linking if you use the module.

Preprocessed source files: F*filename*.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.

Profile data files (.*pdf*)
These are the profile-directed feedback files that the **-qpdf1** compiler option produces. They are used in subsequent compilations to tune optimizations according to actual execution results.

Specifying compiler options

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

You can specify compiler options:

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

When multiple compiler options have been specified, it is possible for option conflicts and incompatibilities to occur. To resolve these conflicts in a consistent fashion, the compiler usually applies the following general priority sequence:

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

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

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

You can also pass compiler options to the linker, assembler, and preprocessor. See XL Fortran Compiler-option reference in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference* for more information about compiler options and how to specify them.

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

After linking, follow the instructions in Chapter 5, “Running XL Fortran programs,” on page 31 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 (file2.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 new objects with existing ones

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

See the Compiler Reference for more information.

Relinking an existing executable file

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

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

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

vi kitchen.f # Fix problem in OVEN subroutine

xlf95 -o newmansion kitchen.f mansion
```

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

Note: 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. 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.

See Linking XL Fortran programs in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference* for more information about linking your programs.

Chapter 5. Running XL Fortran programs

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

You can run a program by entering the name of a program executable file together with any runtime arguments on the command line.

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

Canceling execution

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

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

Setting runtime options

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

For more information on environment variables and how they can affect your applications at runtime, see “Environment variables and XL Fortran” on page 21.

Chapter 6. XL Fortran compiler diagnostic aids

XL Fortran issues diagnostic messages when it encounters problems compiling your application. You can use these messages to help identify and correct such problems.

This section provides a brief overview of the main diagnostics messages offered by XL Fortran. For more information about related compiler options that can help you resolve problems with your application, see Options for error checking and debugging and Options that control listings and messages in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference*.

Note: You might encounter problems when moving from previous versions of XL Fortran to XL Fortran V10. "Avoiding or fixing upgrade problems" on page 16 summarizes these potential problems.

Compilation return codes

At the end of compilation, the compiler sets the return code to zero under any of the following conditions:

- No messages are issued.
- The highest severity level of all errors diagnosed is less than the setting of the **-qhalt** compiler option.

Otherwise, the compiler sets the return code to one of the following values:

Return Code	Error Type
1	An error with a severity level higher than the setting of the **-qhalt** compiler option has occurred.
40	An option error or unrecoverable error has occurred.
41	A configuration file error has occurred.
250	An out-of-memory has occurred. The compiler invocation command cannot allocate any more memory for its use.
251	A signal-received error has occurred. That is, an unrecoverable error or interrupt signal has occurred.
252	A file-not-found error has occurred.
253	An input/output error has occurred - files cannot be read or written to.
254	A fork error has occurred. A new process cannot be created.
255	An error has been detected while the process was running.

Note: Return codes may also be displayed for runtime errors.

XL Fortran compiler listings

Diagnostic information is produced in the output listing according to the settings of the **-qlist**, **-qsource**, **-qxref**, **-qattr**, **-qreport**, and **-qlistopt** compiler options. The **-S** option generates an assembler listing in a separate file.

If the compiler encounters a programming error when compiling an application, the compiler issues a diagnostic message to the standard error device and, if the appropriate compiler options have been selected, to a listing file.

To locate the cause of a problem with the help of a listing, you can refer to:

- 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 precede the section heading so that you can easily locate its beginning:

```
>>>>> section name
```

You can select which sections appear in the listing by specifying the appropriate compiler options. For more information about these options see Options for error checking and debugging and Options that control listings and messages in the *XL Fortran Enterprise Edition V10.1 for AIX Compiler Reference*.

Debugging compiled applications

Specifying the **-g** or **-qlinedebug** compiler options at compile time instructs the XL Fortran compiler to include debugging information in compiled output. You can then use **dbx** or any other symbolic debugger that supports the AIX XCOFF executable format to step through and inspect the behavior of your compiled application.

Chapter 7. XL Fortran runtime environment information

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 Runtime 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 Runtime Environment also produces runtime 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 Runtime Environment. However, static binding is discouraged because it could cause problems when running applications on operating systems that require newer XL Fortran libraries than the one that was statically bound. The compiler defaults to dynamic binding, which is the most flexible solution for running your applications on multiple operating system levels.

The XL Fortran Runtime Environment is upward-compatible. Programs that are compiled and linked with a given level of the runtime environment and a given level of the operating system require the same or higher levels of both the runtime environment and the operating system to run.

External names in the runtime environment

Runtime 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 runtime subprograms called by Fortran object code.

The names of these runtime subprograms are external symbols. When object code that is produced by the XL Fortran compiler calls a runtime 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 runtime subprogram call with the subprogram definition.

You should avoid using names in your XL Fortran program that conflict with names of runtime 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.

External names in the XL Fortran shared libraries

The runtime libraries included in the XL Fortran Runtime 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 runtime libraries, the names of input/output routines in the runtime libraries are prefixed with an underscore(_), or _xl.

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 may contain 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.

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	OS/390
OS/400	POWER	POWER2
POWER5	PowerPC	RS/6000
z/OS	z/VM	

OpenMP is a trademark of the OpenMP Architecture Review Board.

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.

Index

Special characters

Numerics

A

C

D

E

F

H

I

K

L

M

O

P

R

S

T

U

X

Program Number: 5724-M13

SC09-8009-00