IBM XL Fortran for Multicore Acceleration for Linux,
V11.1

Getting Started with XL Fortran

GC23-8524-00

IBM XL Fortran for Multicore Acceleration for Linux,
V11.1

Getting Started with XL Fortran

GC23-8524-00

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 17.

First Edition

This edition applies to IBM XL Fortran for Multicore Acceleration for Linux on System p, V11.1 (Program 5724-T44), and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Contents

About this document

This document contains overview and basic usage information for the IBM® XL Fortran for Multicore Acceleration for Linux®, V11.1 compiler.

Who should read this document

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

How to use this document

XL Fortran provides several compiler invocation commands depending on source code language levels and whether you are compiling for the PowerPC® Processor Unit (PPU) or Synergistic Processor Units (SPUs). However, for convenience, this document uses only the basic **ppuxlf**, and **spuxlf** invocation commands to describe the actions of the Fortran compiler.

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

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

Conventions used in this document

Typographical conventions

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

Table 1. Typographical conventions

Typeface	Indicates	Example
bold	Lowercase commands, executable names, compiler options and directives.	If you specify **-O3**, the compiler assumes **-qhot=level=0**. To prevent all HOT optimizations with **-O3**, you must specify **-qnohot**.

Table 1. Typographical conventions (continued)

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

Syntax diagrams

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

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

  The ▶▶— symbol indicates the beginning of a command, directive, or statement.

  The —▶ symbol indicates that the command, directive, or statement syntax is continued on the next line.

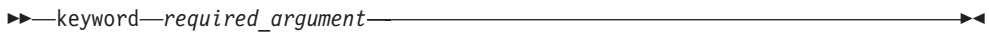
  The ▶— symbol indicates that a command, directive, or statement is continued from the previous line.

  The —▶◀ symbol indicates the end of a command, directive, or statement.

  Fragments, which are diagrams of syntactical units other than complete commands, directives, or statements, start with the |— symbol and end with the —| symbol.

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

  Program units, procedures, constructs, interface blocks and derived-type definitions consist of several individual statements. For such items, a box encloses the syntax representation, and individual syntax diagrams show the required order for the equivalent Fortran statements.
- Required items are shown on the horizontal line (the main path):



- Optional items are shown below the main path:

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

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

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

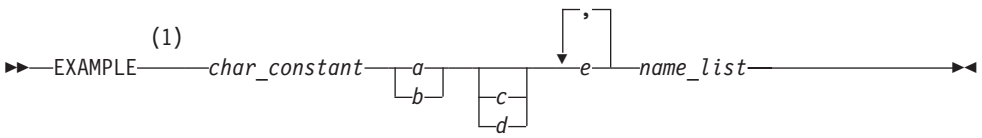
- An arrow returning to the left above the main line (a repeat arrow) indicates that you can make more than one choice from the stacked items or repeat an item. The separator character, if it is other than a blank, is also indicated:

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

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

Sample syntax diagram

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



Notes:

1 IBM extension

Interpret the diagram as follows:

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

Examples

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

Related information

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

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

IBM XL Fortran publications

XL Fortran provides product documentation in the following formats:

- 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 *XL Fortran Installation Guide*.
- PDF documents

  PDF documents are located by default in the doc/en_US/pdf/ directory.

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 for Multicore Acceleration for Linux, V11.1 Installation Guide, GC23-8523-00	install.pdf	Contains information for installing XL Fortran and configuring your environment for basic compilation and program execution.
Getting Started with IBM XL Fortran for Multicore Acceleration for Linux, V11.1, GC23-8524-00	getstart.pdf	Contains an introduction to the XL Fortran product, with information on setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
IBM XL Fortran for Multicore Acceleration for Linux, V11.1 Compiler Reference, SC23-8522-00	cr.pdf	Contains information about the various compiler options and environment variables.
IBM XL Fortran for Multicore Acceleration for Linux, V11.1 Language Reference, SC23-8521-00	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 for Multicore Acceleration for Linux, V11.1 Optimization and Programming Guide, SC23-8525-00	opg.pdf	Contains information on advanced programming topics, such as application porting, interlanguage calls, floating-point operations, input/output, application optimization and parallelization, and the XL Fortran high-performance libraries.

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

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

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

Other IBM publications

- Specifications, white papers, and other technical documents for the Cell Broadband Engine™ architecture are available at http://www.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine.
- The Cell Broadband Engine resource center, at http://www.ibm.com/developerworks/power/cell, is the central repository for technical information, including articles, tutorials, programming guides, and educational resources.

How to send your comments

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

Be sure to include the name of the document, the part number of the document, the version of XL Fortran, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

Chapter 1. Introducing XL Fortran

IBM XL Fortran for Multicore Acceleration for Linux, V11.1 is an advanced, high-performance compiler that can be used for developing complex, computationally intensive programs.

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

Commonality with other IBM compilers

IBM XL Fortran for Multicore Acceleration for Linux, V11.1 is part of a larger family of IBM C, C++, and Fortran compilers.

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

IBM XL Fortran for Multicore Acceleration for Linux, V11.1

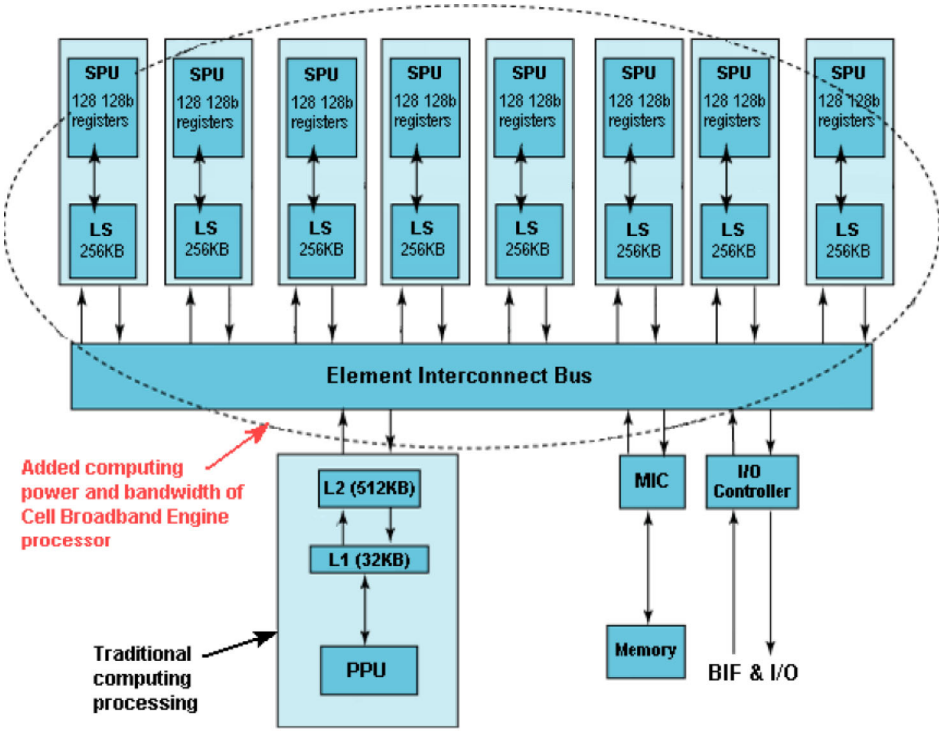
IBM XL Fortran for Multicore Acceleration for Linux, V11.1 is the latest addition to the IBM XL family of compilers. It adopts proven high-performance compiler technologies used in its compiler family predecessors, and adds new features tailored to exploit the unique performance capabilities of processors compliant with the new Cell Broadband Engine architecture.

This version of XL Fortran is a cross-compiler. First, you compile your applications on an IBM System p™ compilation host running Red Hat Enterprise Linux 5.1 (RHEL 5.1). Then you move the executable application produced by the compiler onto a Cell/B.E.™ system also running the RHEL 5.1 Linux distribution. The Cell/B.E. system will be the execution host where you will actually run your compiled application.

About the Cell Broadband Engine architecture

The Cell Broadband Engine architecture specification describes a new single-chip multiprocessor designed to support media-intensive applications.

At the heart of the new multiprocessor is the PowerPC Processor Unit (PPU). The PPU is a 64-bit processor fully compliant with the Power Architecture™ standard, and capable of running both operating systems and applications. The multiprocessor also incorporates a set of eight Synergistic Processor Units (SPUs) into its design. The SPUs are optimized for running computationally intensive applications, operate independently of each other, and can access memory shared between all SPUs and the PPU.



In operation, the PPU runs the operating system and performs high-level application control, while the SPUs divide and perform an application's computational work between them.

For more information on the Cell Broadband Engine architecture, see "*Cell Broadband Engine Architecture from 20,000 feet*" at http://www.ibm.com/developerworks/power/library/pa-cbea.html.

A highly configurable compiler

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

Compiler invocation and linking commands

XL Fortran compiles PPU and SPU program code in separate steps using compiler invocation commands targeted specifically for each type of program code.

Several versions of PPU-specific compiler invocation commands are provided. In most cases, you should compile using the basic **ppuxlf** compiler invocation command, but other variants are also provided to help you meet special compilation needs.

SPU-specific invocation commands are also provided with **spuxlf** and its variants.

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

Compiler options

You can choose from a large selection of compiler options to control compiler behavior. Different categories of options help you to debug your

applications, optimize and tune application performance, select language levels and extensions for compatibility with non-standard features and behaviors supported by other Fortran compilers, and perform many other common tasks that would otherwise require changing the source code.

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

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

Language standards compliance

The compiler supports the following programming language specifications for Fortran:

- ANSI X3.9-1978 (referred to as FORTRAN 77)
- ISO/IEC 1539-1:1991(E) and ANSI X3.198-1992 (referred to as Fortran 90 or F90)
- ISO/IEC 1539-1:1997 (referred to as Fortran 95 or F95)
- Extensions to the Fortran 95 standard:
  - Industry extensions that are found in Fortran products from various compiler vendors
  - Extensions specified in SAA® Fortran
- Most of the Fortran 2003 standard, except for derived type parameters, but including object-oriented programming, as described in "Enhanced support for Fortran 2003"
- Common Fortran language extensions defined by other compiler vendors, in addition to those defined by IBM

In addition to the standardized language levels, XL Fortran supports many industry language extensions, including extensions to support vector programming.

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

Enhanced support for Fortran 2003

XL Fortran supports many Fortran 2003 standard features, including:

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

This is one of the most complete Fortran 2003 implementation currently available, with Derived Type Parameters being the only major feature not yet implemented.

Fortran 2003 compiler invocations and file types

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

- ppuxlf2003, spuxlf2003
- ppuxlf2003_r (for threaded applications)
- ppuf2003, spuf2003

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

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

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

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

Additional Fortran 2003 enhancements

XL Fortran offers enhanced compliance with the Fortran 2003 standard, including:

- Implementation of the full Fortran 2003 object-oriented programming model, including:
  - Type extension
  - Type-bound procedures
  - Type finalization
  - Polymorphism and runtime type determination including the SELECT TYPE construct
  - Abstract and generic interfaces
  - Declaration of abstract types and deferred bindings
  - PASS attribute
- I/O enhancements (PPU only)
  - User-defined derived type I/O
  - New I/O specifiers including SIGN= and DECIMAL= (DC and DP edit descriptors)
  - Asynchronous I/O as defined by Fortran 2003 including the WAIT statement
  - User-specifed control of rounding during format conversion using the ROUND= specifier and new edit descriptors
  - Handling of IEEE infinity and not-a-number in REAL and COMPLEX editing
  - Support of PAD= specifier on INQUIRE operations
- Scoping and data manipulation enhancements
  - Renaming of defined operators on USE statements
  - Fortran 2003 VOLATILE statement
  - MAX, MIN, MAXLOC, MINLOC, MAXVAL, and MINVAL intrinsics for character types
  - COMPLEX literals
  - Pointer assignment and initialization expression enhancements

  - Improved structure constructors
  - Allocatable enhancements including resizing on assignment and MOVE_ALLOC intrinsic
  - Explicit type specification in an array constructor
- Procedure enhancements
  - Generic bindings for interfaces, defined operators, and defined assignment
  - VALUE attribute for characters of length greater than one and derived types with allocatable components
  - Procedure pointers, procedure declaration statement, and procedure pointers as derived type components
  - Generalization of the MODULE PROCEDURE statement
  - Deferred CHARACTER length
- Intrinsic Function Enhancements
  - Allow REAL type for COUNT_RATE argument of SYSTEM_CLOCK
  - Allow boz-literal constants on INT, REAL, CMPLX, and DBL intrinsics
  - Allow a new KIND argument on all instrinsics mandated by Fortran 2003
  - Returning signed zero results from the ATAN2, LOG, and SQRT intrinsics
  - Added SELECTED_CHAR_KIND intrinsic
- Other enhancements
  - Enhanced STOP statement (PPU only)
  - Increased the maximum number of continuation lines

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 compile your application code to a specific language level and warn you if it finds constructs and keywords that do not conform to the specified language level. You can also use the **-qlanglvl** compiler option to specify a given language level, and the compiler will issue warnings if language elements in your program source do not conform to that language level. Additionally, you can name your source files with common filename extensions such as .f77, .f90, f95, or .f03, then use the generic compiler invocations such as **ppuxlf** or **ppuxlf_r** to automatically select the appropriate language-level appropriate to the filename extension.

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

Tools and utilities

new_install
After you install IBM XL Fortran for Multicore Acceleration for Linux, V11.1, running this utility will configure the compiler for use on your system.

xlf_configure
Use this utility to create custom compiler configuration files containing your own custom sets of compiler option default settings.

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

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

Automated program analysis and transformations

Significant performance improvements are possible with relatively little development effort because the compiler is capable of performing sophisticated program analysis and transformation of your program code. For example, the compiler can:

Automatically generate code overlays for the SPUs
Specifying **-qipa=overlay** instructs the compiler to automatically generate code overlays for the SPUs that allow two or more code segments to be loaded at the same physical address as they are needed. This feature lets developers create SPU programs that would otherwise be too large to fit in the local memory store of the SPUs. In addition, the compiler also provides the **-qipa=overlayproc** and **-qipa=nooverlayproc** compiler options to give developers direct control over generation of code overlays on specified procedures.

See **Using automatic code overlays** in the *XL Fortran Optimization and Programming Guide* for more information.

Perform automatic SIMD vectorization of your program code
When the **-qhot=simd** compiler option is in effect, the compiler takes certain operations that are performed in a loop on successive elements of an array, and converts them into a call to a vector instruction. This call calculates several results at one time, which is faster than calculating each result sequentially. Applying this suboption is useful for applications with significant image processing demands.

Not all loops can be successfully vectorized. However, specifying the **-qreport** compiler option together with **-qhot=simd** will cause the compiler to generate diagnostic information that can help you improve the efficiency of your loops.

See for **-qhot** and **-qreport** in the *XL Fortran Compiler Reference* for more information.

Use interprocedural analysis (IPA) to optimize across program files
IPA can result in significant performance improvements. You can specify interprocedural analysis on the compile step only or on both compile and link steps in ″whole program″ mode . Whole program mode expands the scope of optimization to an entire program unit, which can be an executable or shared object.

See **-qipa** in the *XL Fortran 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

XL Fortran also provides optimization features specifically tailored to exploit the unique performance capabilities of Cell Broadband Engine processors, including

specialized data types and highly optimized directives that you can use in your application code to perform common computational needs.

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

- Reduce the number of instructions executed for critical operations.
- Restructure generated object code to make optimal use of the Cell Broadband Engine architecture.
- Improve the usage of the memory subsystem.

Note: For code targeting the SPU, we recommend compiling and linking with the **-O5** or **-qopt=5** compiler options to get the maximum performance from your application.

For more information, see:

- "Optimizing your applications" in the *XL Fortran Optimization and Programming Guide*
- ″Optimization and tuning options″ in the *XL Fortran Compiler Reference*
- To read an article about optimizing performance, search the Power Architecture technical library at www.ibm.com/developerworks/views/power/library.jsp for ″cell broadband tips″.

Diagnostic listings

The compiler output listing can provide important information to help you develop and debug your applications more efficiently.

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

Symbolic debugger support

You can instruct XL Fortran to include debugging information in your compiled objects. That information can be examined by the debuggers provided by the IBM Software Developer Kit (SDK) for Multicore Acceleration V3.0 to help you debug your programs.

Chapter 2. Setting up and customizing XL Fortran

Setting up the IBM XL Fortran for Multicore Acceleration for Linux, V11.1 compiler on your compilation host entails the following main steps:

1. Installing the IBM Software Developer Kit (SDK) for Multicore Acceleration V3.0 development tools on your compilation host.
2. Installing the XL Fortran compiler and runtime environment on your compilation host.

To run completed applications, you will also need to install the SDK and the compiler runtime onto your execution host.

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

Chapter 3. Developing applications with XL Fortran

Basic Fortran application development consists of repeating cycles of editing, compiling, linking, and running.

Note:

1. Before you can use the compiler, you must first ensure that XL Fortran and the IBM Software Developer Kit (SDK) for Multicore Acceleration V3.0 are properly installed and configured. For more information see the *XL Fortran Installation Guide*.
2. To learn about writing Fortran programs, refer to the *XL Fortran Language Reference*.

The compiler phases

A typical compiler invocation executes some or all of the following activities in sequence. For link time optimizations, some activities will be executed more than once during a compilation. As each program runs, the results are sent to the next step in the sequence.

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

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

Editing Fortran source files

To create Fortran source programs, you can use any text editor available to your system. Source programs must be saved using a recognized file name suffix. See the "XL Fortran input and output files" on page 15 for a list of suffixes recognized by XL Fortran.

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

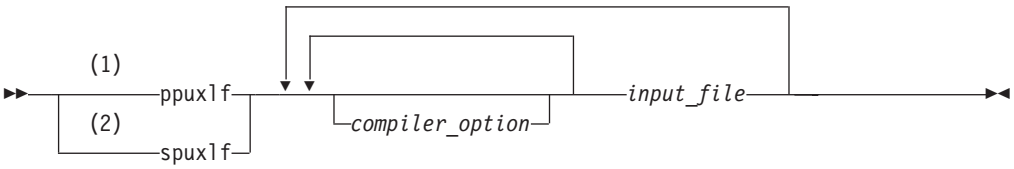
Compiling with XL Fortran

Compiling applications for a Cell/B.E. processor can involve multiple steps, depending on the complexity of your application. For a typical application compiling code for both the PPU and the SPU, you may need to:

1. Compile application code targeted to the PPU.
2. Compile application code targeted to the SPU, then embed the compiled SPU code into PPU code.
3. Perform the final PPU link.

Invoking the compiler

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



Notes:

1 Basic invocation to compile Fortran PPU code.

2 Basic invocation to compile Fortran SPU code.

Compile your application code using a compiler invocation command appropriate to the type of code you are compiling. XL Fortran provides one set of compiler invocation commands for compiling PPU application code, and another set for compiling SPU application code. Application code targeted to the PPU must be compiled and linked using PPU-specific compiler invocations. Similarly, SPU-specific code must be compiled and linked using SPU-specific compiler invocations.

The compiler invocation commands perform all necessary steps to compile Fortran source files and link the object files and libraries into an executable program. Compiled PPU executable objects can be run on the execution host, and will load compiled SPU executable objects at run time as required.

For new application work, you should compile with **ppuxlf**, **spuxlf**, or a thread safe counterpart.

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

Compiling Fortran 95, or Fortran 90 programs

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

Fortran 95	ppuf95, spuf95, ppuxlf95, spuxlf95
Fortran 90	ppuf90, spuf90, ppuxlf90, spuxlf90

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

I/O formats are slightly different between these commands and the other commands. I/O formats for the Fortran 95 compiler invocations are also different from those of Fortran 90 invocations. We recommend that you switch to the Fortran 95 formats for data files whenever possible.

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

For full Fortran 90 compliance:

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

For full Fortran 95 compliance:

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

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

For full Fortran 90 compliance:

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

For full Fortran 95 compliance:

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

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

Compiling Fortran 2003 programs

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

Fortran 2003 ppuf2003, spuf2003, ppuxlf2003, spuxlf2003

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

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

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

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

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

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

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

-qxlf2003 compiler option

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

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

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

Compiling applications that require threadsafe components (PPU only)

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

The generic XL Fortran thread safe compiler invocation is:

- ppuxlf_r, spuxlf_r

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

POSIX Pthreads API support

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

Specifying compiler options

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

You can specify compiler options:

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

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

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

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

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

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

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

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

XL Fortran input and output files

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

Table 3. Input file types

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

Table 4. Output file types

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

Linking your compiled applications with XL Fortran

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

```
ppuxlf file1.f file2.o file3.f
```

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

Compiling and linking in separate steps

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

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

It is usually best to execute the linker through the compiler invocation command, because it passes additional **ppu-ld** or **spu-ld** options and library names to the linker automatically.

Embedding compiled SPU code into compiled PPU code

The following string of compiler commands shows how you might compile an application with both PPU and SPU program segments, and then embed the compiled SPU application code into the compiled PPU application code.

```
spuxlf95 -c spu_file.f
spuxlf95 -o spu_file spu_file.o
ppu32-embedspu spu_file spu_file spu_file-embed.o
ppuxlf95 -c ppu_file.f
ppuxlf95 ppu_file.o spu_file-embed.o
```

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

XL Fortran compiler diagnostic aids

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

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

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

Debugging compiled applications

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 any symbolic debugger to step through and inspect the behavior of your compiled application.

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

Notices

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

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

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

Trademarks and service marks

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

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

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

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

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

Cell Broadband Engine is a trademark of the Sony Corporation and/or the Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom.

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

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

Index

Special characters

A

C

D

E

F

I

L

M

O

P

R

S

T

U

V

Program Number: 5724-T44

GC23-8524-00