



IBM XL C/C++ Compilers for Mac OS X

October 2004

Overview

This paper details the features of the XL C/C++ Advanced Edition for Mac OS X Version 6.0 compiler, including the following highlights:

- C99 features
- GNU compatibility features
- Methods for generating C++ templates
- Hints and tips to improve portability with Mac OS X
- An optimization overview

The XL C/C++ Advanced Edition for Mac OS X compiler is part of a multi-platform XL compiler family derived from a common code base. The common language features simplifies your porting process across operating systems. Table 1 details compiler availability by platform.

Table 1: C/C++ compiler versions

Platform	C/C++
Apple (Mac OS X)	XL C/C++ Advanced Edition V6.0 for Mac OS X
pSeries (AIX)	XL C/C++ Enterprise Edition V7.0 for AIX™
pSeries (Linux) iSeries (Linux)	XL C/C++ Advanced Edition V7.0 for Linux

The Version 6.0 compiler is compatible with Mac OS X 10.2 or 10.3, using the GNU C/C++ compilers V3.3 header files, the corresponding run-time, and the Mac OS X system linker. The compiler offers nearly full source and binary compatibility with GNU 3.3 compilers, and a high degree of source and binary compatibility with the GNU version 3.3 compilers. This compatibility affords you the versatility to build different parts of your application with either the IBM or GNU compiler, and still bind the parts together into a single Mac OS X application. One common use of this functionality is to build an application with XL C/C++ that interoperates with the GNU-built dynamic libraries, without using the library source code. Applications built with this functionality

can integrate with the GNU assembler, and also provide full support for debugging through *gdb*, the GNU debugger.

The XL C/C++ Advanced Edition for Mac OS X compiler also offers support for the XL Fortran Advanced Edition for Mac OS X compiler through interlanguage calls.

XL C/C++ offers developers the opportunity to create and optimize 32-bit applications for the Mac OS X platform. The Mac OS X release brings with it the features, flexibility and refinements that evolved from development on the AIX and Linux platforms. With the addition of language extensions to support the AltiVec Programming Interface and the Apple Velocity Engine, the compiler offers a diverse portfolio of optimization techniques tailored to the PowerPC 970 (G5) architecture, including the 970FX chip. This enables the generation of highly optimized code that exploits the IBM PowerPC architecture within the Power Mac G4 and G5.

The compiler also includes a technical preview of the Objective-C programming language as a basis for implementations using the Cocoa framework. Objective-C is an object-oriented programming language based on standard C language. XL C/C++ Advanced Edition for Mac OS X Version 6.0 also supports OpenMP Version 1.0 for C/C++ as a technical preview.

Users of the Xcode integrated development environment can also invoke XL C/C++ Advanced Edition. New projects created in the Xcode IDE use the native build system. Conversely, Project Builder uses the jam utility, which is similar to the make utility common to other UNIX platforms. While the XL compilers do not support a jam-based build from within Xcode, you can either upgrade an existing Project Builder project to a native Xcode project, or you can use the XL C/C++ compiler with Project Builder to build the project.

Standards Conformance

The C compiler supports the latest ISO/IEC 9899:1999 International Standard, commonly referred to as C99, as well as the previous standard, ISO/IEC 9899:1990 or ISO C89. The C99 standard includes many new features and enhancements to the original ISO/IEC 9899:1990 International Standard (C89), which extends the capability of the C language.

The C++ compiler supports the ISO/IEC 14882:1998 C++ International Standard. C++ also supports a limited form of C99 due to its usefulness in mixed C and C++ code and header file inclusion. The supported C99 features in C++ are *restrict* pointers, function-like macros with variable arguments, and the Pragma operator.

The compilers use the GNU C and C++ headers, and the resulting application is linked with the C and C++ run-time libraries provided with GNU Version 3.3. IBM ships its own implementation of some header files with the product which overrides the corresponding GNU header files. These implementations are functionally equivalent to the corresponding GNU implementations. Other IBM headers are wrappers that include the corresponding GNU header files.

C99 Support

A major revision to the 1989 ANSI/ISO C Standard, called C99, was approved in December, 1998. Most of the new features were already implemented in many C compilers prior to the C99 standard. As such, they are well-tested and have been widely used. With C99 enablement, programmers no longer have to avoid an extension because of portability issues. Furthermore, the C99 standard is upwardly compatible with the previous standards.

The new features can be separated into the following groups:

- New types and keywords
- Floats
- Arrays
- Features from C++
- Convenience features
- Optimization features
- Initialization
- Preprocessor
- Program correctness
- Internationalization
- Library

All of the new C99 functions are supported in the C compiler. The C++ compiler supports restrict pointers, function-like macros and Pragma operators to enable mixed C and C++ compilation and GNU header file inclusion.

Command Line Compatibility and Other Utilities

When you are porting GNU makefiles to XL C/C++, the *gxc* and *gxc++* invocation commands are available to translate a GNU compiler invocation command into the corresponding *xc* or *xc++* command, and invoke the XL C/C++ compiler. This facilitates the transition to XL C/C++ while minimizing the number of changes to makefiles built with a GNU compiler. The *vac_configure* utility can be used to specify the location of the GNU compiler and other information to XL C/C++.

Documentation and Online Help

XL C/C++ supplies comprehensive documentation describing the functionality and capabilities of the compiler as follows:

Table 2: XL C/C++ Documentations

Document	Highlights
Compiler Reference	Contains information about the various compiler options, pragmas, macros, and built-in functions, including those used for parallel processing.
Getting Started	Contains a comprehensive overview of the XL C/C++ Compilers.
Installation Guide	Contains information on installing the XL C/C++ Advanced Edition for Mac OS X Version 6.0 compiler.
Language Reference	Contains information about the C and C++ programming languages, as supported by IBM, including language extensions for portability and conformance to non-proprietary standards.
Programming tasks	Contains information on advanced programming topics such as optimization, data mapping, stream processing, floating point hardware, C++ shared libraries, and framework header files.

These documents are shipped with the product in electronic form (PDF and HTML), or can be downloaded in PDF format from IBM's Web site.

You can view the documentation suite using the Apple Help Center, with a traditional browser, or with a PDF viewer. The Apple Help Viewer provides search capabilities across the XL C/C++ documentation suite. Also included are man pages for all utilities shipped with the compiler, and for compiler invocation commands.

Premier Customer Service

XL C/C++ brings IBM's premier service and support to Mac OS X. The IBM Service and Support organization is made up of a team dedicated to providing you with responsive platform and cross-platform software support. For complex or code-related problems, IBM employs specialized service teams with access to compiler development experts. The vision of IBM Service and Support is to achieve a level of support excellence that exceeds customer expectations and differentiates IBM in the marketplace. You will always have access to the right level of IBM expertise when you need it.

GNU Source Compatibility

The cross-platform portability of gcc and g++ has ensured GNU a place in the Open Source community. GNU has excelled in educational and compiler research arenas as a test bed for new language syntax. IBM compilers are built on a platform of reliability, customer service, and cutting-edge optimization, and are now adapting to gain some of the additional flexibility and portability of the GNU compilers, while still retaining the strengths that have built the XL C/C++ compiler's reputation in the industry.

The C/C++ compilers now support many of the GNU extensions, and allow you to port programs written for GNU compilers to IBM compilers more easily than ever before. The availability of each extension is indicated by a compiler predefined macro of the form `__IBM_feature`, where *feature* is the GNU feature name.

This means that during porting of Open Source code protected by the `__GNUC__` macro, the same code can be easily ported to IBM by adding the `__IBM_feature` attribute. For example, the following code fragment written for GNU:

```
#if defined(__GNUC__)
    extern char *func(const char *__s, int __c)
    __attribute__((__pure__));
#endif
```

will be successfully compiled with the IBM XL C/C++ compiler, if the code is modified as follows:

```
#if defined(__IBM_ATTRIBUTES) || defined(__GNUC__)
    extern char *func(const char *__s, int __c)
    __attribute__((__pure__));
#endif
```

Without GNU Source compatibility support, the code must be modified as follows:

```
#if defined(__GNUC__)
    extern char *func(const char *__s, int __c)
    __attribute__((__pure__));
#else
    extern char *func(const char *__s, int __c);
    #pragma isolated_call(func)
#endif
```

The identifier, `__pure__`, is one of the function attributes supported by the IBM compilers.

Certain GNU language extensions were implemented to enhance source compatibility. Other language extensions that are deemed less common are not supported. Language features that do not affect the code generation are accepted and ignored.

- **Function Attributes**
 - Supported: const, noreturn, pure

- Unsupported: alias, constructor, destructor, weak
- Accepted and Ignored: cdecl, dllexport, dllimport, eightbit_data, exception format, format_arg, function_vector, interrupt, interrupt handler, longcall, model, no_check_memory_usage, no_instrument_function, regparm, stdcall, tiny_data
- **Variable Attributes**
 - Supported: aligned, init_priority (C++), mode, nocommon, packed
 - Unsupported: section, weak
 - Accepted and Ignored: model, section, transparent_union, unused
- **Type Attributes**
 - Accepted and Ignored: aligned packed, transparent_union, unused
- **#warning**
- **#include_next**
- **__extension__** (C)
- **__typeof__**
- **__alignof__**
- **Allow alternate spelling of following keywords**
 - __const__, __volatile__, __signed__, __inline__, and __typeof__
- **Local labels**
- **Address of a label as a constant of type void***
- **Brace-enclosed compound statement inside of parentheses as an expression**
- **Assertions**
 - #assert, #unassert, #cpu, #machine, #system
- **Compound expressions, conditional expressions and casts as lvalues**
- **Declaration of a register variable (global or local) can suggest a preferred register**

GNU Binary Compatibility

The XL C/C++ compilers achieve a high degree of binary compatibility with GNU-built objects, archives, and shared objects. The compiler achieves this by mapping and aligning structures and objects in the same way as GNU compilers, by following the same calling conventions, and by ensuring a close adherence to the Mac OS X ABI.

C++ interoperability is somewhat more difficult to achieve due to different vendors' conventions for name mangling, object model, and exception handling. However, the GNU C++ compiler, since Version 3.2, has adopted a common vendor C++ ABI that defines a way to allow interoperability of C++ object model, name mangling, and exception handling. This common C++ ABI is supported in the XL C++ compilers to enable interoperability with GNU Version 3.3 compilers. Currently, the XL C/C++ compiler Version 6.0 for Mac OS X has been fully tested with GNU C/C++ 3.3 compilers, and offers a high degree of binary compatibility in addition to source compatibility.

OpenMP Support

XL C/C++ Advanced Edition Version 6.0 for Mac OS X includes support for the OpenMP Version 1.0 specification for shared memory parallel programming as a technical preview. OpenMP is an Application Program Interface specification that provides a simple and flexible interface for parallel application development, including pragma directives, data scope attributes, library functions, and environment variables. Applications that conform to the OpenMP specification are easily ported to other platforms that support the specification.

AltiVec Support

XL C/C++ supports the AltiVec programming model through non-orthogonal language extensions. These language extensions match those supported by the Mac OS X GNU compilers. AltiVec data types are referred to as *vector types*. The IBM implementation of the AltiVec Programming Interface specification is an extended syntax that allows type qualifiers and storage class specifiers to precede the keyword vector (or alternately, `__vector`) in a declaration.

The vector keyword is recognized in a declaration context only when used as a type specifier and when you compile the application with `-qaltivec`. The other AltiVec keywords, `pixel` and `bool`, are recognized as valid type specifiers only when preceded by the keyword vector. To ensure maximum portability, use the underscore versions of the specifiers `vector` and `pixel` (`__vector` and `__pixel`) in declarations.

C++ Templates

A template defines a family of related classes or functions, where the parameters in the declaration describe how they are specialized. Templates are generic entities that can be instantiated to create specific user-defined types. The compiler generates new classes or functions when you supply arguments for the parameters. The class or function definition generated from a template with a set of template parameters is called a specialization.

Templates are an area of the C++ language that provides a great deal of flexibility for developers. The ISO C++ standard defines the language facilities and features for templates.

The IBM C++ compiler provides several methods to compile templates:

- Simple layout method. This results in code bloat and longer compile time, but it is easy to use and requires no specific structuring by programmers
- Automatic instantiation using `-qtempinc`. This requires user code structuring but it addresses the long compile time problem inherent in the simple layout method.
- Automatic instantiation using `-qtemplateregistry`. This requires no user code structuring and addresses both the long compile time and code bloat issues.

Each method has its advantages and disadvantages. Finally, the `-qmkshrobj` option is provided to allow building templates in dynamic libraries.

PowerPC Built-in functions

Included in XL C/C++ Advanced Edition Version 6.0 for Mac OS X are a number of built-in functions that map directly to PowerPC hardware instructions. These functions give you access to powerful hardware operations at a source level such as cache prefetching and direct insertion of arithmetic hardware operations. Built-in functions can be used in all the XL C/C++ compilers allowing you to port your code between AIX, Linux, and Mac OS X and still exploit the hardware.

Porting Considerations

Porting an existing AIX or other UNIX^(R)-based application to the Mac OS X platform involves multiple considerations. Often helpful is to filter the diagnostic messages emitted by the compiler to show only those that pertain to portability issues. Compiling with the `-qinfo=por` option enables this filter.

Some steps to perform before porting to the Mac OS X platform are:

- **Check the amount of reliance on GNU C and other language extensions.**
 - An application that conforms strictly to its ISO language specification will be more portable. You may need to revisit code that relies on unsupported extensions.
- **Check how null pointers are dereferenced.**
 - Some errors in the code can go undetected due to operating system-dependent characteristics. This kind of error can occur when the program is ported to another platform. Use the option `-qcheck=nullptr` to help detect such conditions before porting.

The lowest 4K of memory (that is, addresses 0 through 4K-1) is readable and contain zeroes on AIX, but are not readable on the Linux and Mac OS X platforms, and will cause a segmentation violation if accessed. For example,

```
if (strcmp(a, NULL) == 0) ...
```

will result in a segmentation violation on Linux and Mac OS X, but not on AIX.

- **Check alignment.**
 - The supported types of alignment for AIX, Linux, and Mac OS X are not always the same. If you are porting a program that relies on specific values for `-qalign` or `#pragma align`, you may need to change the program. This is especially true if you use or plan to use AltiVec vector types and programming constructs, which impose additional constraints on alignment.
- **Ensure the portability of data structures.**

- If you generate data with an application on one platform and read the data with an application on another platform, the data may have an alignment that is different from that which the reading application expects. To avoid this problem, make sure that you use a platform-neutral mechanism for the layout of data in structures. For example, if you enclose a structure with `#pragma pack(1)` and `#pragma pack(pop)` pair, the alignment will be the same on all platforms. This is especially important if you want data portability between GNU compilers and IBM compilers on AIX and Linux platforms.
- **Use the `gxl` or `gxl++` utility for translating the commands in your makefiles.**
- **Check your global operators.**
 - On Linux or Mac OS X, if the default global operator `new` is called and the allocation request cannot be fulfilled, an exception of type `std::bad_alloc` is thrown. On AIX, the default behavior of the global operator `new` is to return a null pointer if allocation fails.
- **Ensure the portability of applications that use templates.**
 - The C++ compiler provides several different methods of working with template files. Each method has an associated compiler option. The `-qtemplateregistry` compiler option maintains a record of all templates. This method is recommended. An older compiler option, `-qtempinc`, is also provided for applications that you port from another platform. However, on the Mac OS X platform, the compiler option `-qtempinc` is considered deprecated.

Code Optimization

XL C/C++ for Mac OS X provides a portfolio of optimizing transformations tailored to the PowerPC architecture. These transformations can:

- Reduce the number of instructions executed for critical operations.
- Restructure the 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.

Below is a selection of some of the optimizations that the XL C/C++ Advanced Edition for Mac OS X Version 6.0 compiler supports:

- Arch and Tune to target and tune for specific system hardware
- Noopt
- O2
- O3
- O4
- O5
- Interprocedural Analysis: `-qipa`

- High Order loop Transformation: -qhot
- Profile-directed Feedback: -qpdf
- Aliasing control: -qalias
- Strict optimization transformations: -qstrict
- Code size reduction: -qcompact
- Reduces stack size: -qsmallstack
- Inlining control: -qinline
- Independent loop unrolling control: -qunroll
- C++ exception handling support: -qeh
- Stack unwinding control: -qunwind
- Floating-point calculation control: -qfloat

The overall aim of the optimization process is to make your application run faster. Significant performance improvements can be achieved with relatively little development effort if you understand the available controls that affect the transformation of well-written code.

Optimizations are often attempted in the later phases of application development cycles, such as product release builds. If possible, you should test and debug your code without optimization before attempting to optimize it. The optimization process should begin with the most efficient algorithms for your program. To a large extent, compliance with language standards is directly related to the degree to which your code can be successfully optimized. Optimizers can be the ultimate conformance test.

Optimization is controlled by compiler options, directives, and pragmas. However, compiler-friendly programming idioms can be as useful to performance as any of the options or directives. It is no longer necessary or recommended to excessively hand-optimize your code (for example, manually unrolling loops). Unusual constructs can confuse the compiler and make your application difficult to optimize for new machines. It should be noted that not all optimizations are beneficial for all applications. A trade-off usually has to be made between an increase in compile time, accompanied by reduced debugging capability, and the degree of optimization done by the compiler.

Further detail on optimization can be obtained by reading the following white paper:

- Code Optimization with IBM XL Compilers

Architecture and Tuning

The IBM compiler team has extensive experience tuning highly optimized code for the PPCV (G4), PPC970, and the G5 chip.

You can use the -qarch, and -qtune compiler options to optimize the output of the compiler to suit:

- the broadest possible selection of target processors,
- a range of processors within a given processor architecture family,
- a single specific processor.

Generally speaking, the options do the following:

- -qarch selects the general family processor architecture for which instruction code should be generated. Certain -qarch settings produce code that will run *only* on systems that support *all* of the instructions generated by the compiler in response to a chosen -qarch setting.
- -qtune selects the specific processor for which compiler output is optimized. Some -qtune settings can also be specified as -qarch options, in which case they do not also need to be specified as a -qtune option. The -qtune option influences only the performance of the code when running on a particular system but does not determine where the code will run.

Table 3: Options to control code instruction and scheduling for performance

-qarch option	Predefined Macro(s)	Default -qtune setting	Available -qtune setting(s)
ppcv	_ARCH_PPCV	ppc970	auto ppc970 g5
G5	_ARCH_PPCV _ARCH_G5	ppc970	auto ppc970 g5
ppc970	_ARCH_PPCV _ARCH_G5 _ARCH_PPC970	ppc970	auto ppc970 g5

Interprocedural Analysis (IPA)

At O2, and O3, the default optimization is intraprocedural, where each function is optimized individually within a single compilation unit. Starting at O4 and O5, the default optimization is interprocedural, where all functions in an application are analyzed for opportunities to apply optimizations.

In addition to the usual optimizations performed by the low-level optimizing back-end, IPA also performs many optimizations interprocedurally, including:

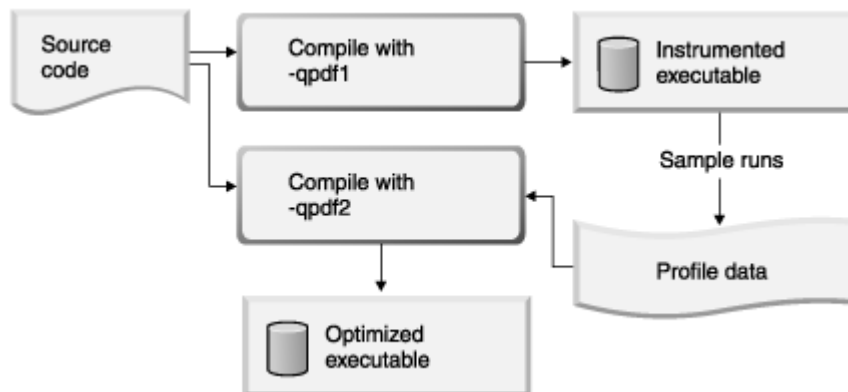
- Inlining across compilation units
- Program partitioning
- Global variables coalescing
- Code straightening
- Dead code elimination
- Constant propagation
- Copy propagation

The cost of improved execution performance is increased compile-time. However, a new option in XL C/C++ Version 6 called `-qipa=threads` can improve compile time by using multi-threaded IPA. Optionally, you can also use `-qipa=threads=N` where N is the number of threads that can be used by IPA.

Profile-Directed Feedback

Near the end of application development, when the code and input data have been stabilized at a high level of optimization, profile-directed feedback can be used to further improve optimization. Using the option `-qpdf1` will instrument the code with special hooks that record execution frequency and branch patterns in a data file when the application is executed. The PDF information is fed back in a second compilation using the option `-qpdf2` and the code will be reordered to ensure the fastest flow path for your particular input data set. This means that the application's input data used to gather PDF information should be representative of the data typically used by the application.

Figure 1: Profile-directed feedback flow chart



Summary

The IBM XL C/C++ Advanced Edition Version 6.0 for Mac OS X compiler is a stable, flexible compiler providing industry leading optimization techniques that can address your compiler needs for everything from small applications, to large, computationally intensive programs.

The extensive cross-platform availability of the IBM XL C/C++ compilers eases the porting process between AIX, z/OS, Linux, and Mac OS X. The extensive standard conformance and GNU compatibilities improve portability of source code from GNU compilers to IBM compilers. The binary compatibility feature allows direct linkage with objects, shared libraries, and archives built by either the GNU or the IBM compilers. This allows the user to take advantage of the features offered by both suites of compiler products. Finally, optimization through chip-specific instruction generation and tuning, parallelization, vectorization, interprocedural analysis, and profile-directed feedback, offers an increase in performance without sacrificing stability or flexibility. Coupled with IBM's excellent service and support, the IBM XL C/C++ Advanced Edition Version 6.0 compiler is a robust and versatile compiler capable of delivering mission critical applications on Mac OS X.

Trial Versions and Purchasing

Trial versions of XL C/C++ compilers can be downloaded for Mac OS X, Linux, and AIX at <http://www.ibm.com/software/awdtools/vacpp/features/xlcpp-mac.html>

Purchasing information for XL C/C++ is also available at that web site. For the Mac OS X version, you can also purchase XL C/C++ from Absoft© Corporation, an IBM OEM partner, at <http://www.absoft.com>.

Contacting IBM

IBM welcomes your comments. You can send them to compinfo@ca.ibm.com or mail them to this address:

XL Compiler Development
Department SQT
Application Development Technology Centre
Software Division Toronto Laboratory IBM Canada Ltd.
8200 Warden Avenue
Markham, Ontario
Canada – L6G 1C7

Copyright Notice

© Copyright IBM Corp. 2004. All Rights Reserved.

IBM is trademark or registered trademark of International Business Machines Corporation in the U.S., other countries or both.