



# Getting Started with VisualAge C++ for AIX

## Introduction, Installation, and Migration Guide

**Note!**

Before using this information and the product it supports, be sure to read the information in "Notices" on page 59.

**First Edition (June, 2002)**

This edition applies to Version 6 Release 0 Modification 0 of IBM VisualAge C++ Professional for AIX (product number 5765-F56) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM® welcomes your comments. You can send them by either of the following methods:

- Internet: [compinfo@ca.ibm.com](mailto:compinfo@ca.ibm.com)

Be sure to include your e-mail address if you want a reply.

- By mail to the following address:

IBM Canada Ltd. Laboratory  
Information Development  
B3/KB7/8200/MKM  
8200 Warden Avenue  
Markham, Ontario, Canada L6G 1C7

Include the title and order number of this book, and the page number or topic related to your comment.

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

## Chapter 1. VisualAge C++ for AIX

<b>Overview</b> . . . . .	<b>1</b>
Command-Line C and C++ Compiler . . . . .	1
Libraries . . . . .	2
Standard C++ Library . . . . .	2
IBM Distributed Debugger . . . . .	2
Other Tools and Utilities . . . . .	3
VisualAge C++ for AIX Documentation and Online Help . . . . .	3
Accessing the Online Documentation . . . . .	4
Accessing Additional Information . . . . .	5

## Chapter 2. New and Special VisualAge C++ for AIX Features

<b>Efficiency, Optimization, and Special Options</b> . . . . .	<b>7</b>
New VisualAge C++ for AIX Options . . . . .	7
New VisualAge C++ for AIX Pragmas . . . . .	8
Other New VisualAge C++ for AIX Features . . . . .	9
New VisualAge C++ for AIX Built-In Functions . . . . .	9
Enhanced Language Level Support . . . . .	11
Conformance to Industry Standards . . . . .	11
ISO/IEC 14882:1998 International Standard Compatibility . . . . .	12
ISO/IEC 9899:1990 International Standard Compatibility . . . . .	12
ISO/IEC 9899:1999 International Standard Support . . . . .	12
Program Parallelism and OpenMP Compliance . . . . .	14
Portability . . . . .	18
Features Related to GNU C/C++ Compilers . . . . .	18
32-Bit and 64-Bit Application Development . . . . .	22
Predefined Macros for C99 Features, Features Related to GNU C/C++ and Other IBM Extensions . . . . .	26
National Language Support . . . . .	27

## Chapter 3. Installing VisualAge C++ for AIX

<b>System Requirements</b> . . . . .	<b>30</b>
Prerequisite Tasks and Conditions . . . . .	31
Installing VisualAge C++ for AIX . . . . .	31
Installing VisualAge C++ for AIX by Selecting Filesets . . . . .	32
Installing All the Components of VisualAge C++ for AIX . . . . .	32
Installing VisualAge C++ for AIX Over a Network . . . . .	32

After Installing VisualAge C++ for AIX . . . . .	33
Installing VisualAge C++ for AIX to a Non-Default Directory . . . . .	33
Enrolling Licenses with LUM . . . . .	35
Enrolling Licenses Using the LUM GUI . . . . .	36
Enrolling LUM Licenses From the Command Line . . . . .	36
Installing Fixes and Upgrades for VisualAge C++ for AIX . . . . .	37
Uninstalling VisualAge C++ for AIX . . . . .	37
VisualAge C++ for AIX Packaging and Filesets . . . . .	37
Filesets Required for C Compiler . . . . .	38
Filesets Required for C Compiler Online Help . . . . .	38
Filesets Required for C++ Compiler . . . . .	39
Filesets Required for C++ Compiler Online Help . . . . .	40
Filesets Required for IBM Distributed Debugger . . . . .	40
Filesets Required to Run LUM and SMIT Client GUIs . . . . .	41
C++ Runtime Filesets Required to Run VisualAge C++ for AIX . . . . .	41
Optional VisualAge C++ for AIX Filesets . . . . .	41
Removed Filesets . . . . .	42
Upgrading the AIX Operating System . . . . .	43

## Chapter 4. Migrating to VisualAge C++ for AIX, Version 6

<b>Incremental Compiler Transition</b> . . . . .	<b>46</b>
IBM Open Class Library Transition . . . . .	47
Visual Builder Transition . . . . .	47
Data Access Builder (DAX) Transition . . . . .	48
Data Access Class Library and Generated Code . . . . .	48
Integrated Development Environment (IDE) Transition . . . . .	49
Performance Analyzer Transition . . . . .	49
Resource Editing Tools Transition . . . . .	49

## Appendix. Equivalent Incremental and Batch Compiler Options

<b>Programming Interface Information</b> . . . . .	<b>61</b>
Trademarks and Service Marks . . . . .	61
Industry Standards . . . . .	61

## Notices

<b>Programming Interface Information</b> . . . . .	<b>61</b>
Trademarks and Service Marks . . . . .	61
Industry Standards . . . . .	61



---

## Chapter 1. VisualAge C++ for AIX Overview

Read this chapter to find out:

- What components and tools are included in IBM VisualAge® C++ Professional for AIX®, Version 6.0?
- How can I access and use the online help? Where can I go for additional information?

IBM VisualAge C++ Professional for AIX, Version 6.0 is a command-line C and C++ compiler for the AIX operating system. Libraries and tools shipped with the compiler help you to effectively create complex programs. The Distributed Debugger shipped with the compiler allows you to visually debug programs running in a client/server environment. Extensive online help ensures that you can easily access the helpful information you need for using VisualAge C++ for AIX.

The VisualAge C++ for AIX compiler builds on the AIX operating system and the PowerPC® architecture to offer many efficient compiler options and built-in functions. It also allows users to compile programs for both 32-bit and 64-bit environments that are compliant with industry standards. In addition, this new version of the compiler offers:

- New compiler options and pragmas for increased efficiency
- Conformance to industry standards
- Efficient template handling
- Support for explicit and automatic parallelization
- Optimizations exploiting the PowerPC architecture
- Support for a subset of features related to GNU C/C++
- Enhanced language level support

For overview information about the components included in VisualAge C++ for AIX, see:

- “Command-Line C and C++ Compiler”
- “Libraries” on page 2
- “IBM Distributed Debugger” on page 2
- “Other Tools and Utilities” on page 3

For additional information, see:

- “Accessing the Online Documentation” on page 4
- “Accessing Additional Information” on page 5

---

### Command-Line C and C++ Compiler

You can use VisualAge C++ for AIX as a C compiler for files with a .c (lowercase c) suffix, or as a C++ compiler for files with a .C (capital C), .cc, .cpp, or .cxx suffix. The compiler processes your text-based program source files to create an executable object module.

In most cases, you should use the **x1C** command to compile your C++ source files, and the **x1c** command to compile C source files. Use the **x1C** command if you have both types of source files.

Compiler options perform a wide variety of functions, such as setting compiler characteristics, describing the object code and compiler output, and performing some preprocessor functions. You can specify compiler options on the command line or in a compiler configuration file. You can also specify a limited number of options in the source file. For more information about new compiler options for VisualAge C++ for AIX, refer to “New VisualAge C++ for AIX Options” on page 7. For detailed information about the various compiler options, refer to the *VisualAge C++ for AIX Compiler Reference*.

#### RELATED REFERENCES

- Compiler Command Line Options in *VisualAge C++ for AIX Compiler Reference*

---

## Libraries

VisualAge C++ for AIX is shipped with the following libraries.

- **c++** Standard C++ Library (including Standard C Library and Standard Template Library) can be used to create code compliant with Standard C++.
- **c++** UNIX<sup>®</sup> Systems Laboratories (USL) I/O Stream Class Library contains stream classes for input and output capabilities for C++.
- **c++** USL Complex Mathematics Class Library contains classes for manipulating complex numbers.
- **c++** C++ Runtime Library contains support routines needed by the compiler.
- **c++** The demangler library provides routines and classes for demangling linkage names created by the C++ compiler.
- SMP Runtime Library supports both explicit and automated parallel processing.
- Memory Debug Runtime is used for diagnosing memory leaks.

### Standard C++ Library

IBM VisualAge C++ Professional for AIX, Version 6.0 is shipped with the Dinkum C++ Library, a conforming implementation of the Standard C++ Library. The Standard C++ Library consists of 51 headers, including 13 headers which constitute the Standard Template Library (STL). In addition, the Standard C++ Library works in conjunction with the 18 headers from the Standard C Library. The functions in these headers perform essential services such as input and output. They also provide efficient implementations of frequently used operations.

#### RELATED CONCEPTS

- C++ Library Overview in *Standard C++ Library Reference*

---

## IBM Distributed Debugger

VisualAge C++ for AIX is shipped with IBM Distributed Debugger.

The IBM Distributed Debugger is a client/server application that enables you to detect and diagnose errors in your programs. This client/server design makes it possible to debug both programs running on systems that are accessible through a network connection and programs that are on your workstation.

The debugger server, also known as a *debug engine*, runs on the same system where the program you want to debug runs. This system can be your workstation or a system accessible through a network. If you debug a program running on your

workstation, you are performing *local debugging*. If you debug a program running on a system accessible through a network connection, you are performing *remote debugging*.

The Distributed Debugger client is a graphical user interface where you can issue commands used by the debug engine to control the execution of your program. For example, you can set breakpoints, step through your code, and examine the contents of variables. The Distributed Debugger user interface lets you debug multiple applications, which might be written in different languages, from a single debugger session. Each program you debug is shown on a separate program page. The type of information that is displayed depends on the debug engine that you are connected to.

By default, the Distributed Debugger is installed to the `/usr/idebug` directory. To start the Distributed Debugger, type `idebug` on the command line.

---

## Other Tools and Utilities

For more information about these tools, commands and utilities, refer to the *VisualAge C++ for AIX Compiler Reference*.

### CreateExportList Command

Creates a file containing a list of all the global symbols found in a given set of object files.

### **C++** c++filt Name Demangling Utility

When VisualAge C++ for AIX compiles a C++ program, it encodes all function names and certain other identifiers to include type and scoping information. This encoding process is called mangling. This utility converts the mangled names to their original source code names.

### **C++** linkxlc Command

Links C++ `.o` and `.a` files. This command is used for linking on systems without VisualAge C++ for AIX compiler installed.

### **C++** makeC++SharedLib Command

Permits the creation of C++ shared libraries on systems on which the VisualAge C++ for AIX compiler is not installed.

### RELATED REFERENCES

- CreateExportList Command in *VisualAge C++ for AIX Programming Topics and Utilities*
- c++filt Name Demangling Utility in *VisualAge C++ for AIX Programming Topics and Utilities*
- linkxlc Command in *VisualAge C++ for AIX Programming Topics and Utilities*
- makeC++SharedLib Command in *VisualAge C++ for AIX Programming Topics and Utilities*

---

## VisualAge C++ for AIX Documentation and Online Help

This section provides an overview of the documentation offered with the product. Information about the compiler, its various tools and utilities, and the C and C++ programming languages is provided through online HTML-based help and Portable Document Format (PDF) publications. You can install the help at the same time as you install VisualAge C++ for AIX.

A plain-text overview of VisualAge C++ for AIX commands and compiler options is also provided. Use your text editor to open `/usr/lib/nls/msg/LANG/vacpp.help` file for C++ compiler help and `/usr/lib/nls/LANG/vac.help` file for C compiler help. *LANG* represents the language and location code. For example, `en_US` is the language and location code for US English (ISO) help.

#### RELATED REFERENCES

- “Accessing the Online Documentation”
- “Accessing Additional Information” on page 5
- “Installing VisualAge C++ for AIX” on page 31
- “VisualAge C++ for AIX Packaging and Filesets” on page 37

## Accessing the Online Documentation

VisualAge C++ for AIX provides you with extensive online documentation. To view the HTML-format online help, you need to have a frames-capable browser such as Netscape Communicator Version 4.04 (or later) installed on your system.

To access the online help for VisualAge C++ for AIX, run the **vacpphelp** command from the `/usr/vacpp/bin` directory.

You can also access the online help from your CDE desktop, by selecting **Application Manager > VisualAge C++ Professional > Help Homepage**.

Much of the online documentation is also available in Adobe Portable Document Format (PDF) format. You can view and print this information using the Adobe Acrobat Reader. If you do not already have the Adobe Acrobat Reader program installed, you can download from the Adobe Web site at <http://www.adobe.com>. The Adobe PDF files are in the `/usr/vacpp/pdf` directory.

The following PDF documents are shipped with VisualAge C++ for AIX and provide detailed explanation of a number of topics covered in this document.

- *VisualAge C++ for AIX C/C++ Language Reference* (`language.pdf`) contains information about C and C++ programming languages, as used by IBM, including the new portability and standardization extensions.
- *VisualAge C++ for AIX Compiler Reference* (`compiler.pdf`) contains information about the various compiler options, pragmas, macros, and built-in functions, including those used for parallel processing.
- *Standard C++ Library Reference* (`stdlib.pdf`) contains detailed information about the Standard C++ Library (including the Standard Template Library) shipped with VisualAge C++ for AIX.
- *VisualAge C++ for AIX Programming Topics and Utilities* (`proguide.pdf`) contains information about programming using VisualAge C++ for AIX not covered in other publications.
- *IBM Open Class Library Transition Guide* (`iocmigr.pdf`) contains migration suggestions for application owners currently using IBM Open Class<sup>®</sup> Library classes.
- *IBM VisualAge C++ Professional for AIX, Version 6.0 License Information* (`license.pdf`) contains important information about the product license.
- *IBM Distributed Debugger* (`debug.pdf`) contains information about the IBM Distributed Debugger tool.

## **Accessing Additional Information**

For the latest information about VisualAge C++ for AIX, visit the IBM VisualAge C++ Web site at <http://www.ibm.com/software/ad/vacpp>.



---

## Chapter 2. New and Special VisualAge C++ for AIX Features

Read this section to find out about the new IBM VisualAge C++ Professional for AIX features, including:

- New compiler options and pragmas for increased efficiency
- Conformance to industry standards
- Efficient template handling
- Support for explicit and automatic parallelization
- Optimizations exploiting the PowerPC architecture
- Support for a subset of features related to GNU C/C++
- Enhanced language level support

---

### Efficiency, Optimization, and Special Options


IBM VisualAge C++ Professional for AIX, Version 6.0 offers new compiler options, pragmas, built-in functions, and other features for ease of development and efficiency of compilation.

#### RELATED REFERENCES

- “New VisualAge C++ for AIX Options”
- “New VisualAge C++ for AIX Pragmas” on page 8
- “Other New VisualAge C++ for AIX Features” on page 9
- “New VisualAge C++ for AIX Built-In Functions” on page 9
- “New Built-In Functions for PowerPC Processors” on page 9

### New VisualAge C++ for AIX Options

New and changed compiler options are described in detail in the *VisualAge C++ for AIX Compiler Reference*.

- The `-qsmp` option to support parallel processing is now available for both C and C++. It is described in “Program Parallelism and OpenMP Compliance” on page 14.
- The `-O4` optimization option is now available for both C and C++ code.
- The `-O5` optimization option is now available for both C and C++ code.
- The `-qipa` option, which activates interprocedural analysis, is now available for both C and C++ code.
- The `-qcache` option, which is used to describe the cache configuration for a specific target execution machine, is now available for both C and C++ code.
- The `-qpdf1` and `-qpdf2` options, which can be used to tune optimizations through Profile-Directed Feedback, are now available for both C and C++ code.
- The `-qalign=bit_packed` suboption, used to specify that the compiler uses `bit_packed` aggregate alignment rules, is now available for both C and C++ code.
-  The new `-qkeepinlines` option instructs the C++ compiler to add the definitions of external inline functions that are not referenced after inlining has occurred to the object file. This option may be required to maintain binary compatibility with builds done with VisualAge C++ for AIX, Version 5.0.

- **c++** The new `-qoldpassbyvalue` option instructs the compiler to mimic IBM C and C++ Compilers for AIX, Version 3.6, so that when a class containing a constant or reference member is passed as a function argument it is not passed by value.
- **c++** The new `-qlanglvl=ansiinit` suboption allows the user to build shared libraries that use the same mechanism for static initialization as used in IBM C and C++ Compilers for AIX, Version 3.6 and earlier versions of the compiler.
- **c++** The new options `-qtemplateregistry` and `-qtemplaterecompile` enable efficient template instantiation.
- The new `-qhot` option determines whether to perform high-order transformations on loops and array language during optimization and whether to pad array dimensions and data objects to avoid cache misses.
- The new `-qarch=pwr4` suboption supports POWER4 architecture.
- The new `-qtune=pwr4` suboption supports POWER4 architecture.
- The new `-qtocmerge` option enables the TOC merge feature in VisualAge C++ for AIX. This feature maps data items to absolute addresses by means of an import file.
- The new `-qreport` option determines whether to produce transformation reports showing how the program is parallelized and how loops are optimized.
- The new `-qipa=threads[=N]` suboption permits the running of a number of parallel threads.
- `-qlargepage` is a new option to support the new POWER4 architecture's support of large 16M pages, in addition to the default 4K pages.
- The new `-qsmallstack` optimization option is intended to reduce the size of the stack frame.
- The new `-qunwind` optimization option allows the compiler to reduce the number of times that registers are saved and restored on the most common paths through a function, if no exception will be thrown through a function.

### Unsupported Compiler Options

The following options are no longer supported in VisualAge C++ for AIX.

- **c** `-qusepcomp` and `-qgenpcomp` options for precompiled headers
- `-qonce` option to avoid including a header file more than once

## New VisualAge C++ for AIX Pragmas

New pragmas are described in detail in the *VisualAge C++ for AIX Compiler Reference*.

- The new `#pragma execution_frequency(very_low)` directive specifies infrequently-executed code.
- The new `#pragma snapshot` directive sets a debugging breakpoint at the point of the pragma, and defines a list of variables to examine when program execution reaches that point.
- The new `#pragma pack` directive allows the users to modify the alignment rule for members of structures.
- The `#pragma unroll` directive, which specifies whether and how the body of a loop can be unrolled, can now be used in both C and C++ code.
- New pragmas for parallel processing are listed in "OpenMP Directives" on page 15.

## Other New VisualAge C++ for AIX Features

- VisualAge C++ for AIX offers more efficient use of templates, including parsing templates at instantiation time, and improved automatic template instantiations.
- The maximum bit-field length is increased to 64 bits.

### RELATED REFERENCES

- Declaring and Using Bit Fields in Structures in *VisualAge C++ for AIX C/C++ Language Reference*

## New VisualAge C++ for AIX Built-In Functions

Several new built-in functions are supported by PowerPC processors. They are listed in “New Built-In Functions for PowerPC Processors” and described in detail in the *VisualAge C++ for AIX Compiler Reference*. OpenMP built-in functions are listed in “OpenMP Library Functions” on page 17 and described in detail in the *VisualAge C++ for AIX Compiler Reference*. In addition, the following new built-in functions were added for VisualAge C++ for AIX:

- The `__prefetch_by_stream` built-in function enables you to load specific data from main memory into the cache.
- The `__iospace_lwsync` and `__isync` built-in function can be used to specify the type of synchronization, and where and how it should take place.

### New Built-In Functions for PowerPC Processors

The following new built-in functions are supported by PowerPC processors.

New built-in functions for PowerPC processors

Function	Description
<code>__clear_lock_up</code>	Clear Lock on UniProcessor systems
<code>__clear_lock_mp</code>	Clear Lock on MultiProcessor systems.
<code>__check_lock_up</code>	Check Lock on UniProcessor systems.
<code>__check_lock_mp</code>	Check Lock on MultiProcessor systems.
<code>__clear_lockd_up</code>	Clear Lock Doubleword on UniProcessor systems.*
<code>__clear_lockd_mp</code>	Clear Lock Doubleword on MultiProcessor systems.*
<code>__check_lockd_up</code>	Check Lock Doubleword on UniProcessor systems.*
<code>__check_lockd_mp</code>	Check Lock Doubleword on MultiProcessor systems.*
<code>__rdlam</code>	Rotate Double Left and AND with Mask.*
<code>__rldimi</code>	Rotate Left Doubleword Immediate then Mask Insert.*
<code>__rlwimi</code>	Rotate Left Word Immediate then Mask Insert.
<code>__rlwnm</code>	Rotate Left Word then AND with Mask.
<code>__rotatel8</code>	Rotate Left Doubleword.*
<code>__rotatel4</code>	Rotate Left Word.
<code>__fctid</code>	Floating Convert to Integer Doubleword.*
<code>__fctidz</code>	Floating Convert to Integer Doubleword with Rounding towards Zero.*
<code>__fctiw</code>	Floating Convert To Integer Word.
<code>__fctiwz</code>	Floating Convert To Integer Word with Rounding towards Zero.
<code>__fcfid</code>	Floating Convert From Integer Doubleword.*

## New built-in functions for PowerPC processors

Function	Description
<code>__stfiw</code>	Store Floating-Point as Integer Word.
<code>__mtfsfi</code>	Move to FPSCR Field Immediate.
<code>__mtfsf</code>	Move to FPSCR Fields.
<code>__mtfsb0</code>	Move to FPSCR Bit 0.
<code>__mtfsb1</code>	Move to FPSCR Bit 1.
<code>__mulhw</code>	Multiply High Word Signed.
<code>__mulhwu</code>	Multiply High Word Unsigned.
<code>__mulhd</code>	Multiply High Doubleword Signed.*
<code>__mulhdu</code>	Multiply High Doubleword Unsigned*
<code>__tw</code>	Trap Word.
<code>__tdw</code>	Trap Doubleword.*
<code>__eieio</code>	Extra name for the existing <code>__iospace_eieio</code> built-in function.
<code>__sync</code>	Extra name for the existing <code>__iospace_sync</code> built-in function.
<code>__lwsync</code>	Extra name for the existing <code>__iospace_lwsync</code> built-in function.
* This built-in function is valid in 64-bit mode only.	

---

## Enhanced Language Level Support

The `-q1anglvl` compiler option is used to specify the supported language level, and therefore affects the way your code is compiled.

For example, to compile your C programs in a way that strictly conforms to the ISO/IEC 9899:1990 International Standard (C89), you need to specify `-q1anglvl=stdc89`. The `stdc89` suboption instructs the compiler to strictly enforce the standard, and not to allow any language extensions.

In order to compile C++ programs in a way that strictly conforms to ISO/IEC 14882:1998 International Standard (Standard C++), you need to specify `-q1anglvl=strict98`.

You can also use extensions to the standard language levels. Extensions that do not interfere with the standard features are called *orthogonal* extensions. For example, when you compile C programs, you can enable extensions that are orthogonal to C89 by specifying `-q1anglvl=extc89`.

Most of the language features described in the ISO/IEC 9899:1999 International Standard (C99) are considered orthogonal extensions to C89.










When you compile C++ programs, you can enable the use of orthogonal extensions by specifying `-q1anglvl=extended`.

In general, a valid program that compiles and runs correctly under a standard language level should continue to compile correctly and run to produce the same result with the orthogonal extensions enabled.

*Non-orthogonal* extensions, on the other hand, can interfere or conflict with aspects of the language as described in one of the international standards. Acceptance of these extensions must be explicitly enabled by individual compiler options.

The main suboptions for the `-qlanglvl` option are listed below.

#### Important `-qlanglvl` Suboptions

<b>-qlanglvl Suboption</b>	<b>Suboption Description</b>
<code>-qlanglvl=stdc99</code>	 Specifies strict conformance to the C99 standard.
<code>-qlanglvl=stdc89</code>	 Specifies strict conformance to the C89 standard.
<code>-qlanglvl=strict98</code>	 Specifies strict conformance to Standard C++. Identical to <code>-qlanglvl=ansi</code> .
<code>-qlanglvl=ansi</code>	 Specifies strict conformance to the C89 standard and enables the <code>-qlonglong</code> compiler option.  Specifies strict conformance to Standard C++. Identical to <code>-qlanglvl=strict98</code> .
<code>-qlanglvl=extc99</code>	 Enables all the orthogonal extensions on top of C99.
<code>-qlanglvl=extc89</code>	 Enables all the orthogonal extensions on top of C89.
<code>-qlanglvl=extended</code>	 Enables all the orthogonal extensions on top of C89 and specifies the <code>-qupconv</code> compiler option.  Enables all the orthogonal extensions on top of Standard C++.

#### RELATED REFERENCES

- `langlvl` in *VisualAge C++ for AIX Compiler Reference*
- `upconv` in *VisualAge C++ for AIX Compiler Reference*
- `longlong` in *VisualAge C++ for AIX Compiler Reference*
- The IBM Language Extensions in *VisualAge C++ for AIX C/C++ Language Reference*
- The IBM C Language Extensions in *VisualAge C++ for AIX C/C++ Language Reference*
- The IBM C++ Language Extensions in *VisualAge C++ for AIX C/C++ Language Reference*

---

## Conformance to Industry Standards

IBM VisualAge C++ Professional for AIX, Version 6.0 builds on accepted industry standards, so your code can be ported easily.

#### RELATED REFERENCES

- “ISO/IEC 14882:1998 International Standard Compatibility” on page 12
- “ISO/IEC 9899:1990 International Standard Compatibility” on page 12
- “ISO/IEC 9899:1999 International Standard Support” on page 12
- “Program Parallelism and OpenMP Compliance” on page 14
- “Enhanced Language Level Support” on page 10

## ISO/IEC 14882:1998 International Standard Compatibility

The ISO/IEC 14882:1998 International Standard (also known as Standard C++) specifies the form and establishes the interpretation of programs written in the C++ programming language. This International Standard is designed to promote the portability of C++ programs among a variety of implementations. For strict conformance to Standard C++, use the `-qlanglvl=strict98` compiler option.

ISO/IEC 14882:1998 is the first formal definition of the C++ language.

## ISO/IEC 9899:1990 International Standard Compatibility

The ISO/IEC 9899:1990 International Standard (also known as C89) specifies the form and establishes the interpretation of programs written in the C programming language. This International Standard is designed to promote the portability of C programs among a variety of implementations. This Standard was amended and corrected by ISO/IEC 9899/COR1:1994, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2:1996. To ensure that VisualAge C++ for AIX compiles your code in a way that enforces the amended and corrected C89 standard, specify the `-qlanglvl=stdc89` compiler option.

## ISO/IEC 9899:1999 International Standard Support

The ISO/IEC 9899:1999 International Standard (C99) is an updated standard for programs written in the C programming language. It is designed to enhance the capability of the C language, provide clarifications and incorporate technical corrections to C89. VisualAge C++ for AIX supports many features of this standard. To ensure that the compiler enforces support for these features, specify the `-qlanglvl=stdc99` compiler option.

Not all runtime functions and facilities required by the ISO/IEC 9899:1999 International Standard are supported on all the operating system levels that can run this version of the compiler. The availability of system header file provides an indication of the support.

### RELATED REFERENCES

- “Major New Features in C99”
- “Changes and Clarifications of C89 Supported in C99” on page 13

### Major New Features in C99

The following C99 features are new in VisualAge C++ for AIX.

#### ISO/IEC 9899:1999 International Standard Extensions to IBM C

New C99 Feature	Related Reference
restrict type qualifier for pointers	The restrict Type Qualifier in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
universal character names	The Unicode Standard in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
predefined identifier <code>__func__</code>	Predefined Identifiers in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
function-like macros with variable and empty arguments	Function-Like Macros in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
<code>_Pragma</code> unary operator	The <code>_Pragma</code> Operator in <i>VisualAge C++ for AIX C/C++ Language Reference</i>

New C99 Feature	Related Reference
variable length array	Arrays in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
static keyword in array index declaration	Arrays in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
complex data type	Complex Types in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
long long int and unsigned long long int types	Integer Variables in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
hexadecimal floating-point constants	Hexadecimal Floating Constants in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
compound literals for aggregate types	Compound Literals in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
designated initializers	Initializers in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
C++ style comments	Comments in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
implicit function declaration not permitted	Function Declarations in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
mixed declarations and code	for Statement in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
_Bool type	Simple Type Specifiers in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
inline function declarations	Inline Functions in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
initializers for aggregates	Initializing Arrays Using Designated Initializers in <i>VisualAge C++ for AIX C/C++ Language Reference</i>

### Changes and Clarifications of C89 Supported in C99

The following C99 features are major changes and clarifications of the C89 standard.

- Flexible array members are allowed. The last member of a structure with two or more members can be declared without the size.
- Declaring implicit int is not supported. All declarations must have a type specifier.
- Multiplicative operators truncate towards zero. Integer division discards the fractional part of the algebraic result.
- Left shift of signed nonnegative integer is now defined.
- Trailing commas are allowed in enumeration specifiers.
- Duplicate type qualifiers are accepted and ignored, unless explicitly specified otherwise.
- A diagnostic message will be issued if a required expression is missing from the return statement.
- Constant expressions evaluated during preprocessing now use long long and unsigned long long data types.
- Empty macro arguments are allowed in function-like macros.
- The maximum value of #line has increased to 2 147 483 647.

## RELATED REFERENCES

- Declaring and Defining a Structure in *VisualAge C++ for AIX C/C++ Language Reference*
- Type Specifiers in *VisualAge C++ for AIX C/C++ Language Reference*
- Division / in *VisualAge C++ for AIX C/C++ Language Reference*
- Bitwise Left and Right Shift <<>> in *VisualAge C++ for AIX C/C++ Language Reference*
- Declaring an Enumeration Data Type in *VisualAge C++ for AIX C/C++ Language Reference*
- The const Type Qualifier in *VisualAge C++ for AIX C/C++ Language Reference*
- return Statement in *VisualAge C++ for AIX C/C++ Language Reference*
- Integer Variables in *VisualAge C++ for AIX C/C++ Language Reference*
- Function-Like Macros in *VisualAge C++ for AIX C/C++ Language Reference*
- Line Control (#line) in *VisualAge C++ for AIX C/C++ Language Reference*

## C99 Features in IBM C++

Some features of the ISO/IEC 9899:1999 International Standard (C99) can also be implemented in C++. Therefore, in addition to the Standard C++ features, VisualAge C++ for AIX supports some C99 features. These extensions are available under the `-qlanglvl=extended` compiler option.

ISO/IEC 9899:1999 International Standard Extensions to IBM C++

C99 Feature	Reference
restrict type qualifier for pointers	The restrict Type Qualifier in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
universal character names	The Unicode Standard in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
predefined identifier <code>__func__</code>	Predefined Identifiers in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
function-like macros with variable and empty arguments	Function-Like Macros in <i>VisualAge C++ for AIX C/C++ Language Reference</i>
<code>_Pragma</code> unary operator	The <code>_Pragma</code> Operator in <i>VisualAge C++ for AIX C/C++ Language Reference</i>

## Program Parallelism and OpenMP Compliance

The OpenMP Application Program Interface (API) supports multi-platform shared-memory parallel programming in C, C++ and Fortran on all architectures, including UNIX platforms and Windows NT<sup>®</sup> platforms. OpenMP is a portable, scalable model that gives programmers a simple and flexible interface for developing shared-memory parallel applications. It is jointly defined by a group of major computer hardware and software vendors, including IBM.

VisualAge C++ for AIX is OpenMP Specification 1.0 compliant, since it recognizes and preserves the semantics of all its elements. The directives, library functions, and environment variables described below allow you to create and manage parallel programs while permitting portability.

The C compiler has had OpenMP Specification 1.0 support since IBM VisualAge C++ Professional for AIX, Version 5. In this release, OpenMP Specification 1.0 support is extended to the C++ compiler.

To enable OpenMP parallel processing, you must specify the `-qsmp` compiler option.

- To choose automated parallelism, specify `-qsmp` or `-qsmp=auto`. This suboption enables the compiler to perform *implicit parallelism*, in addition to recognizing and implementing any OpenMP directives, library functions, and environment variables included in the program.
- To choose strict compliance to the OpenMP Specification 1.0, specify `-qsmp=omp`. This suboption ensures that the compiler implements only the OpenMP directives, library functions, and environment variables specified in the code. It does not perform any additional automated parallel processing.

For details about parallel processing and the VisualAge C++ for AIX implementation of the OpenMP Specification 1.0, refer to the *VisualAge C++ for AIX Compiler Reference*.

For more information about the OpenMP Specification, visit the OpenMP Web site at <http://www.openmp.org>.

#### RELATED REFERENCES

- “OpenMP Directives”
- “OpenMP Data Scope Attribute Clauses” on page 17
- “OpenMP Library Functions” on page 17
- “OpenMP Environment Variables” on page 18
- “OpenMP Implementation Defined Behavior” on page 18
- Pragmas to Control Parallel Processing in *VisualAge C++ for AIX Compiler Reference*

#### RELATED CONCEPTS

- Program Parallelization in *VisualAge C++ for AIX Compiler Reference*

### OpenMP Directives

Each directive starts with `#pragma omp`, to reduce the potential for conflict with other pragma directives.

OpenMP directives in VisualAge C++ for AIX

Directive Name	Directive Description
<code>parallel</code>	The <code>parallel</code> directive defines a <i>parallel region</i> , which is a region of the program that is to be executed by multiple threads in parallel.
<code>for</code>	The <code>for</code> directive identifies an iterative work-sharing construct that specifies a region in which the iterations of the associated loop should be executed in parallel. The iterations of the <code>for</code> loop are distributed across threads that already exist.
<code>sections</code>	The <code>sections</code> directive identifies a non-iterative work-sharing construct that specifies a set of constructs that are to be divided among threads in a team. Each section is executed once by a thread in the team.
<code>single</code>	The <code>single</code> directive identifies a construct that specifies that the associated structured block is executed by only one thread in the team (not necessarily the master thread).

## OpenMP directives in VisualAge C++ for AIX

Directive Name	Directive Description
parallel for	The parallel for directive is a shortcut form for a parallel region that contains a single for directive. The semantics are identical to explicitly specifying a parallel directive immediately followed by a for directive.
parallel sections	The parallel sections directive provides a shortcut form for specifying a parallel region containing a single sections directive. The semantics are identical to explicitly specifying a parallel directive immediately followed by a sections directive.
master	The master directive identifies a construct that specifies a structured block that is executed by the master thread of the team.
critical	The critical directive identifies a construct that restricts execution of the associated structured block to a single thread at a time. An optional <i>name</i> may be used to identify the critical region. A thread waits at the beginning of a critical region until no other thread is executing a critical region with the same name. All unnamed critical directives map to the same unspecified name.
barrier	The barrier directive synchronizes all the threads in a team. When encountered, each thread waits until all of the others have reached this point. After all threads have encountered the barrier, each thread begins executing the statements after the barrier directive in parallel.
atomic	The atomic directive identifies a specific memory location that must be updated atomically and not be exposed to multiple, simultaneous writing threads.
flush	The flush directive identifies a point at which the compiler ensures that all threads in a parallel region have the same view of specified objects in memory.
ordered	The ordered directive identifies a structured block of code that must be executed in sequential order.
threadprivate	The threadprivate directive declares file-scope or namespace-scope variables to be private to a thread, but file-scope visible within the thread.

The C compiler recognizes the following additional directives used for program parallelism. They are not part of the OpenMP Specification 1.0, and are not recognized by the C++ compiler.

- #pragma ibm critical
- #pragma ibm independent\_calls
- #pragma ibm independent\_loop
- #pragma ibm iterations
- #pragma ibm parallel\_loop
- #pragma ibm permutation
- #pragma ibm schedule
- #pragma ibm sequential\_loop

## OpenMP Data Scope Attribute Clauses

Clauses may be specified on the directives to control the scope attributes of variables for the duration of the parallel or work-sharing constructs.

OpenMP data scope attribute clauses in VisualAge C++ for AIX

Data Scope Attribute Clause Name	Data Scope Attribute Clause Description
private	The private clause declares the variables in the list to be private to each thread in a team.
firstprivate	The firstprivate clause provides a superset of the functionality provided by the private clause.
lastprivate	The lastprivate clause provides a superset of the functionality provided by the private clause.
shared	The shared clause shares variables that appear in the list among all the threads in a team. All threads within a team access the same storage area for shared variables.
reduction	The reduction clause performs a reduction on the scalar variables that appear in list, with a specified operator.
default	The default clause allows the user to affect the data scope attributes of variables.

## OpenMP Library Functions

OpenMP runtime library functions are included in the header `<omp.h>`. They include *execution environment functions* that can be used to control and query the parallel execution environment, and *lock functions* that can be used to synchronize access to data.

OpenMP runtime library functions in VisualAge C++ for AIX

Runtime Library Function Name	Runtime Library Function Description
omp_set_num_threads	Sets the number of threads to use for subsequent parallel regions.
omp_get_num_threads	Returns the number of threads currently in the team executing the parallel region from which it is called.
omp_get_max_threads	Returns the maximum value that can be returned by calls to <code>omp_get_num_threads</code> .
omp_get_thread_num	Returns the thread number, within its team, of the thread executing the function. The master thread of the team is thread 0.
omp_get_num_procs	Returns the maximum number of processors that could be assigned to the program.
omp_in_parallel	Returns non-zero if it is called within the dynamic extent of a parallel region executing in parallel; otherwise, it returns 0.
omp_set_dynamic	Enables or disables dynamic adjustment of the number of threads available for execution of parallel regions.
omp_get_dynamic	Returns non-zero if dynamic thread adjustment is enabled and returns 0 otherwise.
omp_set_nested	Enables or disables nested parallelism.

OpenMP runtime library functions in VisualAge C++ for AIX

Runtime Library Function Name	Runtime Library Function Description
omp_get_nested	Returns non-zero if nested parallelism is enabled and 0 if it is disabled.
omp_init_lock	Initializes a simple lock.
omp_destroy_lock	Removes a simple lock.
omp_set_lock	Waits until a simple lock is available.
omp_unset_lock	Releases a simple lock.
omp_test_lock	Tests a simple lock.
omp_init_nest_lock	Initializes a nestable lock.
omp_destroy_nest_lock	Removes a nestable lock.
omp_set_nest_lock	Waits until a nestable lock is available.
omp_unset_nest_lock	Releases a nestable lock.
omp_test_nest_lock	Tests a nestable lock.

## OpenMP Environment Variables

OpenMP environment variables control the execution of parallel code. The names of environmental variables must always be in upper case, while their values are not case sensitive.

OpenMP environment variables in VisualAge C++ for AIX

Environment Variable Name	Environment Variable Description
OMP_SCHEDULE	Sets the runtime schedule type and chunk size.
OMP_NUM_THREADS	Sets the number of threads to use during execution.
OMP_DYNAMIC	Enables or disables dynamic adjustment of the number of threads.
OMP_NESTED	Enables or disables nested parallelism.

## OpenMP Implementation Defined Behavior

The following information is not specified in the standard. Each implementation of the standard may have its own implementation-defined values.

### Conditional Compilation:

The `_OPENMP` macro is defined to 199810.

### Scheduling:

The `schedule` clause specifies how iterations of the `for` loop are divided among threads of the team. The possible OpenMP standard values are `static`, `dynamic`, `guided`, and `runtime`. In addition, there is an IBM C extension value `affinity`. In the absence of an explicitly defined `schedule` clause, the default schedule for VisualAge C++ for AIX is `runtime`.

---

## Portability

### Features Related to GNU C/C++ Compilers

To ease porting of code written for GNU C/C++, some GNU C/C++ features are supported in VisualAge C++ for AIX. If you have code written for GNU C/C++, you can use this section to help you port your code to VisualAge C++ for AIX.

To use *supported* features with your C code, specify one of `-qlanglvl=extended`, `-qlanglvl=extc89`, or `-qlanglvl=extc99`. To use these features with your C++ code, specify the `-qlanglvl=extended` option.

The compiler recognizes *accept/ignore* features as acceptable programming keywords, but does not support them.

Compiling source code that uses these features under a strict `-qlanglvl` suboption will result in error messages.

For more information on GNU C/C++, see <http://www.gnu.org/software/gcc/>.

#### RELATED REFERENCES

- “GCC Function Attributes”
- “GCC Variable Attributes” on page 20
- “GNU C/C++ Type Attributes” on page 20
- “GNU C/C++ Assertions” on page 21
- “Other Extensions Related to GNU C/C++” on page 21

### GCC Function Attributes

Use the keyword `__attribute__` to specify special attributes when making a function declaration. This keyword is followed by an attribute specification inside double parentheses.

GCC function attribute compatibility with VisualAge C++ for AIX

Function Attribute	Behavior
alias	accept/ignore
cdecl	accept/ignore
const	supported
constructor	accept/ignore
destructor	accept/ignore
dllexport	accept/ignore
dllimport	accept/ignore
eightbit_data	accept/ignore
exception	accept/ignore
format	accept/ignore
format_arg	accept/ignore
function_vector	accept/ignore
interrupt	accept/ignore
interrupt_handler	accept/ignore
longcall	accept/ignore
model	accept/ignore
no_check_memory_usage	accept/ignore
no_instrument_function	accept/ignore
noreturn	supported
pure	supported
regparm	accept/ignore

#### GCC function attribute compatibility with VisualAge C++ for AIX

Function Attribute	Behavior
section	accept/ignore
stdcall	accept/ignore
tiny_data	accept/ignore


#### RELATED REFERENCES

- Function Attributes in *VisualAge C++ for AIX C/C++ Language Reference*

### GCC Variable Attributes

Use the keyword `__attribute__` to specify special attributes of variables or structure fields. This keyword is followed by an attribute specification inside double parentheses.

#### GCC variable attribute compatibility with VisualAge C++ for AIX

Variable Attribute	Behavior
aligned	supported
 init_priority	accept/ignore
mode	supported
model	accept/ignore
noccommon	accept/ignore
packed	supported
section	accept/ignore
transparent_union	accept/ignore
unused	accept/ignore

#### RELATED REFERENCES

- Variable Attributes in *VisualAge C++ for AIX C/C++ Language Reference*

### GNU C/C++ Type Attributes

Use the keyword `__attribute__` to specify special attributes of struct and union types when you define these types. This keyword is followed by an attribute specification inside double parentheses. While type attributes are not currently supported in VisualAge C++ for AIX, some type attributes are accepted and ignored by the compiler.

#### GCC type attribute compatibility with VisualAge C++ for AIX

Type Attribute	Behavior
aligned	accept/ignore
packed	accept/ignore
transparent_union	accept/ignore
unused	accept/ignore

#### RELATED REFERENCES

- Type Attributes in *VisualAge C++ for AIX C/C++ Language Reference*

## GNU C/C++ Assertions

Use *assertions* to test what sort of computer or system the compiled program will run on. The assertions `#cpu`, `#machine`, and `#system` are predefined. You can also define assertions by with preprocessing directives `#assert` and `#unassert`.

GNU C/C++ assertions in VisualAge C++ for AIX

GCC Assertions	Behavior
<code>#assert</code>	supported
<code>#unassert</code>	supported
<code>#cpu</code>	supported
<code>#machine</code>	supported
<code>#system</code>	supported possible values are <code>aix</code> and <code>unix</code>

## Other Extensions Related to GNU C/C++

The following features related to GNU C/C++ are supported under some non-strict `-qlanglvl` suboptions.

- Use directive `#warning` to cause the preprocessor to issue a warning and continue processing.
- Use directive `#include_next` to specify inclusion of the next header file in a directory after the current one.
- Local labels can be declared at start of each lexical block.
- Refer to the type of an expression with the `typeof` keyword.
- Use compound expressions, conditional expressions, and casts as lvalues.
- Use keyword `__alignof__` to inquire about variable alignment, or the alignment usually required by a type.
- Use alternate spelling of these keywords: `__const__`, `__volatile__`, `__signed__`, `__inline__`, and `__typeof__`.

Under some `-qlanglvl` suboptions, VisualAge C++ for AIX recognizes the syntax of the following features. However, the semantics of these features is not supported.

- You can define a global register variable, or a local register variable residing in specified registers.
- Write `__extension__` before an expression containing a GNU C/C++ extension to avoid receiving a warning.

Many existing extensions to IBM C/C++ are also supported in GNU C/C++.

### RELATED REFERENCES

- Preprocessor Warning Directive (`#warning`) in *VisualAge C++ for AIX C/C++ Language Reference*
- Specialized File Inclusion in *VisualAge C++ for AIX C/C++ Language Reference*
- Locally Declared Labels in *VisualAge C++ for AIX C/C++ Language Reference*
- `typeof` Operator in *VisualAge C++ for AIX C/C++ Language Reference*
- Lvalues and Rvalues in *VisualAge C++ for AIX C/C++ Language Reference*
- Keywords in *VisualAge C++ for AIX C/C++ Language Reference*

## 32–Bit and 64–Bit Application Development

You can use VisualAge C++ for AIX to develop both 32-bit and 64-bit applications. This section outlines various portability considerations in moving C and C++ programs from 32-bit to 64-bit mode.

### RELATED REFERENCES

- “Constants”
- “Assignment of Long Variables to Integers and Pointers” on page 23
- “Undeclared Functions” on page 24
- “Structure Sizes and Alignments” on page 24
- “Bit Fields” on page 24
- “Miscellaneous Issues” on page 25
- “Interlanguage Calls with Fortran” on page 25

### Constants

The limits of constants change. This table shows changed items in the limits.h header file, their hexadecimal value, and decimal equivalent. The equation gives an idea of how to construct these values.

Changes in Limits of Constants

Type	Hexadecimal	Equation	Decimal
signed long min (LONG_MIN)	0x8000000000000000L	$-(2^{63})$	-9,223,372,036,854,775,808
signed long max (LONG_MAX)	0x7FFFFFFFFFFFFFFFL	$2^{63}-1$	+9,223,372,036,854,775,807
unsigned long max (ULONG_MAX)	0xFFFFFFFFFFFFFFFFUL	$2^{64}-1$	+18,446,744,073,709,551,616

In C and C++, type identification of constants follows explicit rules. However, programs that use constants exceeding the limit (relying on a 2’s complement representation) will experience unexpected results in the 64-bit mode. This is especially true of hexadecimal constants and unsuffixed constants, which are more likely to be extended into the 64-bit long type.

Problematic behaviors will generally occur at boundary areas such as:

- constant  $\geq$  UINT\_MAX
- constant  $<$  INT\_MIN
- constant  $>$  INT\_MAX

Some examples of undesirable boundary side effects are:

Undesireable Boundary Side Effects Examples

Constant assigned to long	32 bit mode	64 bit mode
-2,147,483,649 (INT_MIN-1)	+2,147,483,647	-2,147,483,649
+2,147,483,648 (INT_MAX+1)	-2,147,483,648	+2,147,483,648
+4,294,496,726 (UINT_MAX+1)	0	+4,294,967,296
0xFFFFFFFF (UINT_MAX)	-1	+4,294,496,295
0x100000000 (UINT_MAX+1)	0	+4,294,967,296

### Undesireable Boundary Side Effects Examples

Constant assigned to long	32 bit mode	64 bit mode
0xFFFFFFFFFFFFFFFF (ULONG_MAX)	-1	-1

Currently, the compiler gives out of range warning messages when attempting to assign a value larger than the designated range into a long type. This warning message may not appear for every case.

When you bit left-shift a 32-bit constant and assign it into a long type, signed values are sign-extended and unsigned values are zero-extended. The examples in the table below show the effects of performing a bit-shift on both 32- and 64-bit constants, using the following code segment:

```
long l=constantL<<1;
```

### Constant Value Bit-Shift Examples

Initial Constant Value	Constant Value after Bit-Shift	
	32-bit	64-bit
0x7FFFFFFFL (INT_MAX)	0xFFFFFFFFE	0xFFFFFFFFE
0x80000000L (INT_MIN)	0	0x100000000
0xFFFFFFFFL (UINT_MAX)	0xFFFFFFFFE	0x1FFFFFFFFE

Unsuffixed constants can lead to type ambiguity that can impact other parts of your program, such as the result of **sizeof** operations. For example, in 32-bit mode the compiler types a number like 4294967295 (UINT\_MAX) as an unsigned long. In 64-bit mode, this same number becomes a signed long. To avoid this possibility, explicitly add a suffix to all constants that have the potential of impacting constant assignment or expression evaluation in other parts of your program. The fix for the above case is to write the number as 4294967295U. This forces the compiler to always see that constant as an unsigned int regardless of compiler mode.

### Assignment of Long Variables to Integers and Pointers

Using int and long types in expressions and assignments can lead to implicit conversion through promotions and demotions, or explicit conversions through assignments and argument passing. The following should be avoided:

- Using integer and long types interchangeably, leading to truncation of significant digits or unexpected results.
- Passing long arguments to functions expecting type int.
- Exchanging pointers and int types, causing segmentation faults.
- Passing pointers to a function expecting an int type, resulting in truncation.
- Assignment of long types to float, causing possible loss of accuracy.

Assigning a long constant to an integer will cause truncation without warning. For example:

```
int i;  
long l=2147483648; /* INT_MAX+1*/  
i=l;
```

What will be the value of i? INT\_MAX+1 is 2147483647+1 (0x80000000), which becomes INT\_MIN when assigned into a signed type. Truncation occurs because the highest bit is treated as a sign bit. The rule here is that there will be a loss of significant digits.

Similar problems occur when passing constants directly to functions, and in functions that return long types. Making explicit use of the L and UL suffix will avoid most, but not all, problems. Alternately, you can avoid accidental conversions by using explicit prototyping. Another good practice is to avoid implicit type conversion by using explicit type casting to change types.

### **Undeclared Functions**

Any function that returns a pointer should be explicitly declared when compiling in 64-bit mode. Otherwise, the compiler will assume the function returns an int and truncates the resulting pointer, even if you were to assign it into a valid pointer.

Code constructs such as the following are valid in 32-bit mode.

```
a=(char *) calloc(25);
```

However, in 64-bit mode, the pointer a will be silently truncated. Even type casting will not prevent the truncation: the memory allocated by calloc() was already truncated after the return.

The fix in this case would be to include the appropriate header file, which is `stdlib.h`.

### **Structure Sizes and Alignments**

Structures might present porting problems.

The 64-bit specification changes the size, member and structure alignment of all structures that are recompiled in 64-bit mode. Structures with long types and pointers will generally change size and alignment in 64-bit mode. Some structures may not change in size because they happen to fall on an exact 8-byte boundary even in 32-bit mode.

Sharing data structures between 32- and 64-bit processes is no longer possible unless the structure is devoid of pointer and long types. Unions that attempt to share long and int types, or overlay pointers onto int types will now be aligned differently, or be corrupted. In general, all but the simplest structures must be checked for alignment and size dependencies.

The alignment for `-qalign=full`, `power` or `natural` changes for 64-bit mode. Structure members are aligned on their natural boundaries. Long types and pointer types are word-aligned in 32-bit mode, and doubleword aligned in 64-bit mode. Additional spaces could be used for padding members.

The alignment for `-qalign=twobyte` and `-qalign=mac68k` are not supported in 64-bit mode.

Structures are aligned according to the strictest aligned member. This remains unchanged from 32-bit mode. Because of the padding introduced by the member alignment, structure alignment may not be exactly the same as in the 32-bit mode. This is especially important when you have arrays of structures which contain pointer or long types. The member alignment will change, most likely leading to the structure alignment to change to doubleword alignment (if there are no long types, double types, and long double types).

### **Bit Fields**

Structure bit fields are limited to 32 bits in 32-bit mode, and can be of type signed int, unsigned int or plain int. Bit fields are packed into the current word. Adjacent bit fields that cross a word boundary will start at storage unit. This storage unit is

a word in power and full alignment, halfword in the mac68k and twobyte alignment, and byte in the packed alignment.

In 32-bit mode, non-integer bit fields are tolerated (but not respected) only in the C extended language level. C++ tolerates non-integer bit fields in any language level.

64-bit bit fields are supported in 64-bit mode.

### Miscellaneous Issues

- The sizeof operator will now return size\_t which is an unsigned long.
- The length of the integer required to hold the difference between two pointers is ptrdiff\_t, and is a signed long type.
- Masks will generally lead to different results when compiled in 64-bit mode from their 32-bit mode behavior.
- Many include files have pointers and structures in them, and their inclusion in 64-bit mode will change the size of your data section even if your program does not use structures and pointers explicitly.
- \_\_int64 is a long type in 64-bit mode, but will look like a long long type in 32-bit mode. \_\_int64 types can participate in promotion rules and arithmetic conversion when in 64-bit mode. When in 32-bit mode, these types can not participate in the usual arithmetic conversions.
- In 64-bit mode, member values in a structure passed by value to a va\_arg argument may not be accessed properly if the size of the structure is not a multiple of 8-bytes. This is a known limitation of the operating system.
- In 64-bit extended mode, zero-extension from unsigned int to an unsigned long preserves the bit pattern. For example, zero-extending an unsigned int with value 0xFFFF FFFF (large negative value) results in an unsigned long with value 0x0000 0000 FFFF FFFF (large positive value).

### Interlanguage Calls with Fortran

A significant number of applications use C, C++, and Fortran together, by calling each other or sharing files. Such applications are among the early candidates for porting to 64-bit platforms for their abilities to solve larger mathematical models. Experience shows that it is easier to modify data sizes and types on the C side than the Fortran side of such applications. The following table lists C and C++ types and the equivalent Fortran types in the different modes.

Equivalent C/C++ and Fortran Data Types

C/C++ type	32-bit	64-bit
int	INTEGER	INTEGER
unsigned int	LOGICAL	LOGICAL
signed long	INTEGER	INTEGER*8
unsigned long	LOGICAL	LOGICAL*8
pointer	INTEGER	INTEGER*8






























In 64-bit mode, Fortran has a POINTER\*8 that is 8 bytes in length as compared to POINTER which is 4 bytes in length.

## Predefined Macros for C99 Features, Features Related to GNU C/C++ and Other IBM Extensions

The following macros are defined have the value of 1 if the listed feature is supported under the specified `qlanglvl` suboption. If the feature is not supported, then the macro is undefined. For description of the features, see “ISO/IEC 9899:1999 International Standard Support” on page 12 and “Features Related to GNU C/C++ Compilers” on page 18.

All predefined macros are protected.

### Predefined Macros for C99 Features, Features Related to GNU C/C++ and Other IBM Extensions

Feature	Predefined Macro Name	Supported in <code>-qlanglvl</code> Suboption
flexible array member	<code>__C99_FLEXIBLE_ARRAY_MEMBER</code>	 <code>stdc99, extc99</code>
duplicated type qualifier	<code>__C99_DUP_TYPE_QUALIFIER</code>	 <code>stdc99, extc99, extc89, extended</code>
new limit for <code>#line</code>	<code>__C99_MAX_LINE_NUMBER</code>	 <code>stdc99, extc99, extc89, extended</code>
<code>_Bool</code> type	<code>__C99_BOOL</code>	 <code>stdc99, extc99, extc89, extended</code>
long long type	<code>__C99_LLONG</code>	 <code>stdc99, extc99</code>
inline function specifier	<code>__C99_INLINE</code>	 <code>stdc99, extc99, extc89, extended</code>
restrict qualifier	<code>__C99_RESTRICT</code>	 <code>stdc99, extc99 -qkeyword=restrict</code>  <code>-qkeyword=restrict</code>
static keyword in array declaration	<code>__C99_STATIC_ARRAY_SIZE</code>	 <code>stdc99, extc99, extc89, extended</code>
universal character name	<code>__C99_UCN</code>	 <code>stdc99, extc99, extc89, extended</code>  <code>extended</code>
variable length arrays	<code>__C99_VAR_LEN_ARRAY</code>	 <code>stdc99, extc99, extc89, extended</code>
<code>__func__</code> keyword	<code>__C99_FUNC__</code>	 <code>stdc99, extc99, extc89, extended</code>  <code>extended</code>
hexadecimal floating constants	<code>__C99_HEX_FLOAT_CONST</code>	 <code>stdc99, extc99, extc89, extended</code>
C++ style comments	<code>__C99_CPLUSCMT</code>	 <code>stdc99, extc99</code>
compound literals	<code>__C99_COMPOUND_LITERAL</code>	 <code>stdc99, extc99, extc89, extended</code>
designated initialization	<code>__C99_DESIGNATED_INITIALIZER</code>	 <code>stdc99, extc99, extc89, extended</code>
mixed declaration and code	<code>__C99_MIXED_DECL_AND_CODE</code>	 <code>stdc99, extc99, extc89, extended</code>
function-like macros with variable arguments	<code>__C99_MACRO_WITH_VA_ARGS</code>	 <code>stdc99, extc99, extc89, extended</code>  <code>extended</code>
standard pragmas	<code>__C99_STD_PRAGMAS</code>	 <code>stdc99, extc99, extc89, extended</code>
<code>_Pragma</code> operator	<code>__C99_PRAGMA_OPERATOR</code>	 <code>stdc99, extc99, extc89, extended</code>  <code>extended</code>
complex type	<code>__C99_COMPLEX</code>	 <code>stdc99, extc99, extc89, extended</code>
type generic macros <code>&lt;tgmath.h&gt;</code>	<code>__C99_TGMATH</code>	 <code>stdc99, extc99, extc89, extended</code>
implicit function declaration not supported	<code>__C99_REQUIRE_FUNC_DECL</code>	 <code>stdc99</code>
concatenation of wide string and non-wide string	<code>__C99_MIXED_STRING_CONCAT</code>	 <code>stdc99, extc99, extc89, extended</code>
subscripting in non-lvalue arrays	<code>__C99_NON_LVALUE_ARRAY_SUB</code>	 <code>stdc99, extc99, extc89, extended</code>
non-constant array initializers	<code>__C99_NON_CONST_AGGR_INITIALIZER</code>	 <code>stdc99, extc99, extc89, extended</code>

## Predefined Macros for C99 Features, Features Related to GNU C/C++ and Other IBM Extensions

Feature	Predefined Macro Name	Supported in -qlanglvl Suboption
local labels	<code>__IBM_LOCAL_LABEL</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div> <div style="display: flex; align-items: center; margin-top: 5px;"> <div style="margin-right: 5px;">▶ C++</div> <div>extended</div> </div>
labels as values	<code>__IBM_LABEL_VALUE</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div>
typeof	<code>__IBM_TYPEOF__</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div><code>__typeof__</code>: extc99, extc89, extended; <code>typeof</code>: -qkeyword=typeof</div> </div> <div style="display: flex; align-items: center; margin-top: 5px;"> <div style="margin-right: 5px;">▶ C++</div> <div><code>__typeof__</code>: extended; <code>typeof</code>: -qkeyword=typeof</div> </div>
generalized lvalues	<code>__IBM_GENERALIZED_LVALUE</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div>
function attributes	<code>__IBM_ATTRIBUTES</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div> <div style="display: flex; align-items: center; margin-top: 5px;"> <div style="margin-right: 5px;">▶ C++</div> <div>extended</div> </div>
dollar signs in identifiers	<code>__IBM_DOLLAR_IN_ID</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div>
variable attributes	<code>__IBM_ATTRIBUTES</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div> <div style="display: flex; align-items: center; margin-top: 5px;"> <div style="margin-right: 5px;">▶ C++</div> <div>extended</div> </div>
type attributes	<code>__IBM_ATTRIBUTES</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div>
<code>__alignof__</code>	<code>__IBM_ALIGNOF__</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div> <div style="display: flex; align-items: center; margin-top: 5px;"> <div style="margin-right: 5px;">▶ C++</div> <div>extended</div> </div>
inline functions	<code>__IBM_GCC_INLINE__</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div><code>__inline__</code>: extc99, extc89, extended -qgcc_inline can also control both <code>__inline__</code> and <code>inline</code> keywords</div> </div>
explicit register variables	<code>__IBM_REGISTER_VARS</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div>
alternate keywords	<code>__IBM_ALTERNATE_KEYWORDS</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div>
<code>__extension__</code>	<code>__IBM_EXTENSION_KEYWORD</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div> <div style="display: flex; align-items: center; margin-top: 5px;"> <div style="margin-right: 5px;">▶ C++</div> <div>extended</div> </div>
empty macro arguments	<code>__C99_EMPTY_MACRO_ARGUMENTS</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>stdc99, extc99, extc89, extended</div> </div>
<code>#assert</code> , <code>#unassert</code> , <code>#cpu</code> , <code>#machine</code> , <code>#system</code>	<code>__IBM_PP_PREDICATE</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div>
<code>#warning</code>	<code>__IBM_PP_WARNING</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div>
<code>#include_next</code>	<code>__IBM_INCLUDE_NEXT</code>	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">▶ C</div> <div>extc99, extc89, extended</div> </div> <div style="display: flex; align-items: center; margin-top: 5px;"> <div style="margin-right: 5px;">▶ C++</div> <div>extended</div> </div>

## National Language Support

VisualAge C++ for AIX has support that enables you to create international applications, including Unicode support, multibyte character support, and bidirectional support.

### RELATED REFERENCES

- The Unicode Standard in *VisualAge C++ for AIX C/C++ Language Reference*
- National Languages Support in VisualAge C++ in *VisualAge C++ for AIX Compiler Reference*



---

## Chapter 3. Installing VisualAge C++ for AIX

This section contains information about the prerequisites, requirements, and environmental considerations for product installation.

VisualAge C++ for AIX uses License Use Management (LUM) to control the licenses for the product. LUM is a component of the base operating system in AIX Version 4.3 or higher. You do not need to install LUM, but you might need to configure it before installing VisualAge C++ for AIX. You may also want to install the latest version of LUM. It is available for download at <http://www.ibm.com/software/is/lum>. It is also provided on the VisualAge C++ for AIX CD in the /lum directory.

After installing VisualAge C++ for AIX, you need to enroll your license with License Use Management. LUM documentation, including configuration instructions, is available in PDF at <http://www.ibm.com/software/is/lum/library.html>.

You can also find simplified configuration instructions in the README.password file in /usr/vac for the C compiler and in /usr/vacpp for the C++ compiler.

You need to install and configure LUM only once. If LUM is already installed and configured on your system, you can skip to “Installing VisualAge C++ for AIX” on page 31.

If this is the first time you are installing VisualAge C++ for AIX, follow these steps:

1. Install VisualAge C++ for AIX.
2. Enroll licenses with LUM.

If you have a previous version of VisualAge C++ for AIX installed, follow these steps:

1. Uninstall the previous version of VisualAge C++ for AIX.
2. Install VisualAge C++ for AIX.
3. Enroll licenses with LUM.

### RELATED REFERENCES

- “System Requirements” on page 30
- “Prerequisite Tasks and Conditions” on page 30
- “Installing VisualAge C++ for AIX” on page 31
- “Enrolling Licenses with LUM” on page 35
- “Installing Fixes and Upgrades for VisualAge C++ for AIX” on page 37
- “Uninstalling VisualAge C++ for AIX” on page 37
- “VisualAge C++ for AIX Packaging and Filesets” on page 37
- “Upgrading the AIX Operating System” on page 43

---

## System Requirements

### Display

SVGA 800x600 (1024x764 recommended)<sup>1</sup>

### CD-ROM drive

Required

### Mouse or pointing device

Required<sup>1</sup>

### Memory (RAM)

96 MB minimum (128 MB or higher recommended)

### Disk space

Up to 525 MB<sup>1</sup>

### Operating System

IBM AIX Version 4.3.3 or higher

### To access the online help

To access the online help, you need to use the Common Desktop Environment (CDE), Adobe Acrobat Reader to view PDF files, and a frames-capable browser such as Netscape Communicator Version 4.04 (or higher).

---

## Prerequisite Tasks and Conditions

Because of the complexity of the VisualAge C++ for AIX product, not every prerequisite has been listed in these instructions. Use the preview option to verify and display the required software for your choice of components.

### Checking for Required Filesets:

The following items *must* be installed on your system.

Fileset Name	Fileset Description
bos.adt.include	Base Application Development Include Files
bos.adt.lib	Base Application Development Libraries
bos.adt.libm	Base Application Development Math Libraries
bos.net.ncs	Base Network Computing Services
ifor_ls.compat	License Use Management Version 4 Compatibility
ifor_ls.base	License Use Management Version 4 Base

Use the following command to determine if these items have been installed:

```
lslpp -h bos.adt.include bos.adt.lib bos.adt.libm \  
bos.net.ncs ifor_ls.compat ifor_ls.base
```

Refer to the *AIX Installation Guide* if you need to install these products.

### Checking for Other Filesets:

---

1. Required for the Distributed Debugger, LUM GUI, and online help in HTML and PDF formats.

The following optional items are prerequisites for some components.

Fileset Name	Fileset Description
X11.base.rte	AIXwindows Runtime Environment. Install this if you want to invoke the GUI versions of the LUM tools, or if you want to have desktop icons for VisualAge C++ for AIX help.
bos.rte.libpthreads	pthread Library. Required for threaded applications.
ipfx.rte	IPF/X Runtime Support. Install this if you want to invoke the GUI versions of the LUM tools.
ifor_ls.base.gui and ifor_ls.client.gui	License Use Runtime Filesets. Install this if you want to invoke the GUI versions of the LUM tools.

Use the following command to determine if these items have been installed:

```
lslpp -h X11.base.rte bos.rte.libpthreads \  
ipfx.rte ifor_ls.base.gui ifor_ls.client.gui
```

Refer to the *AIX Installation Guide* if you need to install these products.

### Verifying Space Requirements:

To verify the amount of space needed for the installation, choose the following settings on the SMIT Install Software Products at Latest Level panel before you install VisualAge C++ for AIX:

- PREVIEW only? (install operation will NOT occur)
- VERIFY install and check file sizes

The system makes additional resource checks during installation. If you want, you can also specify the following installation option in SMIT:

```
EXTEND file systems if space needed
```

---

## Installing VisualAge C++ for AIX

You can install VisualAge C++ for AIX in one of two ways:

- Select the individual filesets you want to install on your machine. This allows you to control which components of VisualAge C++ for AIX are installed. For more information about the filesets included on the product CD, see “VisualAge C++ for AIX Packaging and Filesets” on page 37.
- Install all the filesets found on a CD, as long as they are relevant to your operating system. This option installs all the components found on a CD, giving you the most complete installation of VisualAge C++ for AIX possible.

After you have installed VisualAge C++ for AIX, you need to enroll your license for the product before using it. For help on installing your license, see “Enrolling Licenses with LUM” on page 35.

**Note:** If you are upgrading an existing installation of VisualAge C++ for AIX, you should uninstall your existing version of VisualAge C++ for AIX before installing IBM VisualAge C++ Professional for AIX, Version 6.0. See “Uninstalling VisualAge C++ for AIX” on page 37 for help on uninstalling the product.

You must have root user access to install VisualAge C++ for AIX.

## Installing VisualAge C++ for AIX by Selecting Filesets

1. Insert the CD into your CD-ROM device
2. At a command prompt, enter `smit install_latest`.
3. Press PF4 or click **List** to display a list of devices.
4. Select the CD-ROM device, then click **OK**.
5. Press PF4 or click **List** to select the filesets that you want to install.

Some of the VisualAge C++ for AIX filesets are specific to the AIX version and language environments on your machine. If you select filesets that do not match your version of AIX or your language environments, you will receive a failed install message. We recommend that you read the fileset descriptions closely.

**Note:** When fileset names differ only by the AIX version that supports them, you should install only the fileset supported by the version of AIX running on your machine. If your machine is running AIX Version 4.3.3, choose *this.aix43.fileset*. If your machine is running AIX 5L™, choose *this.aix50.fileset*.

6. Follow the on-screen instructions to complete this installation.

### RELATED REFERENCES

- “After Installing VisualAge C++ for AIX” on page 33
- “Installing VisualAge C++ for AIX to a Non-Default Directory” on page 33

## Installing All the Components of VisualAge C++ for AIX

1. Insert the CD into your CD-ROM device.
2. At a command prompt, enter `smit install_latest`.
3. Press PF4 or click **List** to display a list of devices.
4. Select the CD-ROM device, then click **OK**.
5. Click **OK**.
6. Follow the on-screen instructions to complete this installation.

Some of the VisualAge C++ for AIX filesets are specific to the AIX version and language environments on your machine. You may receive an error message at the end of the install process.

If you receive an error message at the end of the installation process, check the names and descriptions of the filesets that did not install. File sets not intended for your version of AIX are expected failures. In addition, file sets that require language environments that are not available on your system are also expected failures.

### RELATED REFERENCES

- “After Installing VisualAge C++ for AIX” on page 33
- “Installing VisualAge C++ for AIX to a Non-Default Directory” on page 33

## Installing VisualAge C++ for AIX Over a Network

If you have a network server installed, you can install VisualAge C++ for AIX over a network.

1. At a command prompt, enter `smit install_latest`.

2. In the **Device** field, enter the directory on the client that corresponds to the installation source for on the network server, then click **OK**.
3. (Optional) Press PF4 or click **List** to select the filesets that you want to install. Some of the VisualAge C++ for AIX filesets are specific to the AIX version and language environments on your machine. If you select filesets that do not match your version of AIX or your language environments, you will receive a failed install message. We recommend that you read the fileset descriptions closely.

**Note:** When fileset names differ only by the AIX version that supports them, you should install only the fileset supported by the version of AIX running on your machine. If your machine is running AIX Version 4.3.3, choose *this.aix43.fileset*. If your machine is running AIX 5L, choose *this.aix50.fileset*.

4. Follow the on-screen instructions to complete this installation.

You can also use the Network Install Manager (NIM) to perform network installs. Refer to the *AIX Network Installation Management Guide and Reference* for more information.

#### RELATED REFERENCES

- “After Installing VisualAge C++ for AIX”
- “Installing VisualAge C++ for AIX to a Non-Default Directory”

## After Installing VisualAge C++ for AIX

After you have installed VisualAge C++ for AIX, you need to enroll your license for the product before using it. For help on installing your license, see “Enrolling Licenses with LUM” on page 35.

VisualAge C++ for AIX is not automatically installed in `/usr/bin`. To invoke the compiler without having to specify the full path, do one of the following steps:

- Create symbolic links for the specific driver contained in `/usr/vacpp/bin` and `/usr/vac/bin` to `/usr/bin`.
- Add `/usr/vacpp/bin` and `/usr/vac/bin` to your path environment variable.

If you use a method other than the AIX **installp** command or the SMIT utility to install VisualAge C++ for AIX (such as the non-default install script), the location of the drivers will vary from the default locations above.

## Installing VisualAge C++ for AIX to a Non-Default Directory

If you use the AIX **installp** command, or the SMIT installation utility, VisualAge C++ for AIX is normally installed to `/usr/vacpp/` directory. You can install VisualAge C++ for AIX in a non-default directory using a Perl script provided with the product. This can be helpful if, for example, you want to be able to run two versions of VisualAge C++ for AIX.

The non-default install Perl script `vacppndi` is provided in fileset `vacpp.ndi`. Install this fileset first, and then use the script to install VisualAge C++ for AIX to an alternate target directory.

You may choose to install just the compiler filesets, or the compiler, the sample files, and the online help. You cannot install IBM Distributed Debugger using the `vacppndi` script.

**Note:** You should only use `vacppndi` script to install VisualAge C++ for AIX if you are an expert AIX user. To avoid unexpected behavior during installation, do not modify this script.

In order to run the Perl script, you must have the Perl Version 5 runtime environment `perl.rte` installed on your computer. This fileset is shipped with the AIX 5.1 base operating system. To download and install the Perl runtime environment on your AIX 4.3.3 system, refer to the Comprehensive Perl Archive Network Web site at <http://www.cpan.org>.

For detailed information about the `vacppndi` script, type `/usr/vacpp/bin/vacppndi -h` on the command line and press Enter.

**To install VisualAge C++ for AIX to a non-default directory:**

1. Navigate to the `/usr/vacpp/bin/` directory.
2. On the command line, type

```
perl vacppndi -d source_path [-e logfile_name] \  
                [-m] [-p target_directory]
```

  - Use `source_path` to specify the directory where the VisualAge C++ for AIX filesets are located. This path may also be a mounted CD-ROM drive.
  - Use the `-e logfile_name` option to specify the name and location of the installation log file. Otherwise, the installation log file `vacppndi.log` will be stored in your working directory.
  - Use the `-m` option to install just the C and C++ compiler files. If you do not specify this option, compiler files, sample files, and online help files will be installed.
  - Use the `-p target_directory` option to specify the location where the filesets should be copied and expanded. Otherwise, the files will be copied to the `/usr/vacpp/` directory. If the target directory exists already, you will receive an error message and the installation will stop.
3. Press Enter.

**To install a PTF (fix) on the non-default version of VisualAge C++ for AIX:**

1. Create a text file listing the PTF files you want to install. This text file should contain a name of a single PTF file on each line.
2. Navigate to the `/usr/vacpp/bin` directory.
3. On the command line, type

```
perl vacppndi -d source_path [-e logfile_name] \  
                [-p target_directory] -u ptf_names
```

  - Use `source_path` to specify the directory where the PTF filesets are located.
  - Use the `-e logfile_name` option to specify the name and location of the installation log file. Otherwise, the installation log file `vacppndi.log` will be stored in your working directory.
  - Use the `-p target_directory` option to specify the location where the filesets should be copied and expanded. Otherwise, the files will be copied to the `/usr/vacpp/` directory.
  - Use `ptf_names` to specify the names of PTF packages you want to install.
4. Press Enter.

If you install VisualAge C++ for AIX using the `vacppndi` script, you will not be able to use AIX tools to uninstall it, or to determine which version of the compiler components is installed. To uninstall the non-default version of VisualAge C++ for

AIX, delete the directory specified as *target\_directory* during the installation. To determine which version of each fileset is installed, review the *.vmf\_history* text file, which is created in the *target\_directory* during installation of the compiler and any fixes.

#### RELATED REFERENCES

- “VisualAge C++ for AIX Packaging and Filesets” on page 37

---

## Enrolling Licenses with LUM

Before starting to use VisualAge C++ for AIX, you must enroll the appropriate license certificate, using the License Use Management (LUM) software. Three LUM license certificates are provided with each compiler product: a concurrent nodelock license certificate, a concurrent network license certificate, and a simple nodelock license certificate. You should enroll the appropriate certificate for the type of license server you have configured. If you plan to program using both C and C++ languages, you will need to enroll a license certificate for each language.

Before enrolling a LUM license certificate, ensure that you have installed, configured, and started LUM on your machine.

LUM license file names and locations for VisualAge C++ for AIX

Compiler Language	C	C++
Directory	/usr/vac	/usr/vacpp
Concurrent nodelock LUM license file name	cforaix_cn.lic	vacpp_cn.lic
Concurrent network LUM license file name	cforaix_c.lic	vacpp_c.lic
Simple nodelock LUM license file name	cforaix_n.lic	vacpp_n.lic

Before you enroll concurrent nodelock and concurrent network LUM license certificates, make sure you have the correct licenses for the authorized users of this program.

#### Note:

IBM VisualAge C++ Professional for AIX, Version 6.0 is licensed based on a charge unit of authorized users for each physical machine running the program.

An authorized user is an individual or specific named user authorized to have access to the program or any portion of the program on a physical machine. The Proof of Entitlement for this program is evidence of your authorization. The number of authorized users (individual or specific named users) allowed to access the IBM VisualAge C++ Professional for AIX, Version 6.0 program is determined by the number of authorizations the customer acquires.

Only an authorized user may have access to the program or any portion of the program.

#### RELATED REFERENCES

- “Enrolling Licenses Using the LUM GUI” on page 36
- “Enrolling LUM Licenses From the Command Line” on page 36

## Enrolling Licenses Using the LUM GUI

The LUM Basic License Tool runs either from a GUI or a command line interface. You must have root user access to enroll your VisualAge C++ for AIX license with LUM.

To enroll a license certificate using the LUM Basic License Tool GUI:

1. To invoke the LUM Basic License Tool, run the **i4blt** command.
  - On AIX 4.3.3, the **i4blt** command is in the `/var/ifor/` directory.
  - On AIX 5.1 and higher, the **i4blt** command is in the `/usr/opt/ls/os/aix/bin/` directory.
2. Select **Products > Enroll Product** from the main menu. The Enroll Product dialog box appears.
3. Click **Import**. The Import dialog box opens.
4. In the **Filter** field, enter the directory name listed in “Enrolling Licenses with LUM” on page 35 and click **Filter**.
5. Select the correct license certificate file name from the **Files** field, then click **OK**. Information about your VisualAge C++ for AIX license should now be displayed in the Enroll Product window.
6. Click **OK**. The Enroll Licenses window opens.
7. (Optional) Fill in the Administrator Information part of the Enroll Licenses window.
8. Fill in the number of valid purchased licenses of the product in the Product Information part of the Enroll Licenses window.
9. Click **OK**. The product should be successfully enrolled.
10. To exit the LUM Basic License Tool, select **Products > Exit**.
11. If you have enrolled concurrent network licenses, you must distribute the licenses before starting to use VisualAge C++ for AIX. For instructions on how to distribute licenses, see the LUM documentation at <http://www.ibm.com/software/is/lum/library.html>. You can also find simplified instructions for distributing licenses in the `README.password` file in `/usr/vac` for the C compiler and `/usr/vacpp` for the C++ compiler.

## Enrolling LUM Licenses From the Command Line

To enroll a license certificate using the LUM Basic License Tool command line interface:

1. Extract the **i4blt** command from the top of the correct product license file.
2. Replace `number_of_lics` in the command with the number of valid purchased licenses of the product.
3. (Optional) Replace `admin_name` in the command with the name of the administrator.
4. Invoke the updated command from the correct directory.
  - On AIX 4.3.3, the **i4blt** command is in the `/var/ifor/` directory.
  - On AIX 5.1 and higher, the **i4blt** command is in the `/usr/opt/ls/os/aix/bin/` directory.

The product should be successfully enrolled.

5. If you have enrolled concurrent network licenses, you must distribute the licenses before starting to use VisualAge C++ for AIX. For instructions on how to distribute licenses, see the LUM documentation at <http://www.ibm.com/software/is/lum/library.html>. You can also find

simplified instructions for distributing licenses in the README.password file in /usr/vac for the C compiler and /usr/vacpp for the C++ compiler.

---

## Installing Fixes and Upgrades for VisualAge C++ for AIX

You can download the latest fixes and upgrades for VisualAge C++ for AIX from the **Support** section of the IBM VisualAge C++ for AIX web site at <http://www.ibm.com/software/ad/vacpp>.

You must have root user access to install fixes for VisualAge C++ for AIX.

1. Save the downloaded fix files to a directory accessible by your client machine.
2. Remove the .toc file in the download directory after each download to create a new table of contents.
3. (Optional) To view a list of problems resolved by this update, type `smit install_list_problems` on the command line and press Enter.
4. On the command line, type `smit install_latest` and press Enter.
5. Press PF4 to display a list of devices.
6. Specify the download directory as the **INPUT device / directory for software**, then press Enter.
7. To install the full product, press Enter. You can also press PF4 to select the filesets you want to install.
8. Follow the instructions to complete the installation.

---

## Uninstalling VisualAge C++ for AIX

You must have root user access to uninstall this product.

1. At the command line, type `smit install_remove` and press Enter. The Remove Installed Software window opens.
2. On the **SOFTWARE Name** line, press PF4. A list of the available software filesets appears.
3. Select all VisualAge C++ for AIX filesets, then press Enter.

Some filesets may not uninstall if they are required by other, installed products. See “VisualAge C++ for AIX Packaging and Filesets” for details about filesets included with VisualAge C++ for AIX.

---

## VisualAge C++ for AIX Packaging and Filesets

If you do not want to install all available components, you may choose which filesets to install. In addition, you may specify that any fileset that is a prerequisite to a fileset you selected be installed automatically.

When fileset names differ only by the AIX version that supports them, you should install only the fileset supported by the version of AIX running on your machine. If your machine is running AIX Version 4.3.3, choose *this.aix43.fileset*. If your machine is running AIX 5L, choose *this.aix50.fileset*.

When fileset names differ only by the language code, you should install only the filesets relevant to your desired language and location. The LANG environment variable determines which message catalogs are used. If a catalog for the corresponding LANG cannot be found, English message catalogs are used instead.

**Note:** In the tables below, *LANG* represents one of the national language codes. For example, US English (ISO) C compiler messages are in *vac.msg.en\_US.C* fileset.

**Note:** All the filesets required for the C compiler are also required for the C++ compiler.

**RELATED REFERENCES**

- “Filesets Required for C Compiler”
- “Filesets Required for C Compiler Online Help”
- “Filesets Required for C++ Compiler” on page 39
- “Filesets Required for C++ Compiler Online Help” on page 40
- “Filesets Required for IBM Distributed Debugger” on page 40
- “Filesets Required to Run LUM and SMIT Client GUIs” on page 41
- “C++ Runtime Filesets Required to Run VisualAge C++ for AIX” on page 41
- “Optional VisualAge C++ for AIX Filesets” on page 41
- “Removed Filesets” on page 42

## Filesets Required for C Compiler

The following filesets are included in the C compiler.

Filesets Required for the Command-Line C Compiler

Fileset Name	Fileset Description
<i>vac.C</i>	C for AIX compiler
<i>vac.C.readme.ibm</i>	C for AIX additional information
<i>vac.lic</i>	C for AIX LUM License Files
<i>vac.msg.LANG.C</i>	C for AIX compiler messages
<i>xlopt.lib</i>	XLOPT Optimization Library
<i>xlopt.rte</i>	XLOPT Optimization Runtime
<i>xlopt.tools</i>	XLOPT Optimization Tools
<i>xlsmp.msg.LANG.rte</i>	XL SMP Runtime Messages
<i>xlsmp.rte</i>	XL SMP Runtime Library
<i>memdbg.adt</i>	User Heap/Memory Debug Toolkit
<i>memdbg.aix43.adt</i>	User Heap/Memory Debug Toolkit for AIX 4.3
<i>memdbg.aix50.adt</i>	User Heap/Memory Debug Toolkit for AIX 5.1
<i>memdbg.msg.LANG</i>	User Heap/Memory Debug Messages

## Filesets Required for C Compiler Online Help

The following filesets are included in C compiler online help.

Filesets Required for C Compiler Online Help

Fileset Name	Fileset Description
<i>vac.Dt.common</i>	C for AIX Desktop Integration Common Files
<i>vac.Dt.help</i>	C for AIX Help Desktop Integration
<i>vac.html.LANG.C</i>	C for AIX Compiler HTML Documentation

### Filesets Required for C Compiler Online Help

Fileset Name	Fileset Description
vac.html.LANG.search	C for AIX Compiler Documentation Search
vac.html.DBCS.search	C for AIX Compiler Documentation Search Double Byte Common Files
vac.html.SBCS.search	C for AIX Compiler Documentation Search Single Byte Common Files
vac.html.common.search	C for AIX Compiler Documentation Search Common Files
vac.loc.LANG.Dt.help	C for AIX Help Desktop
vac.pdf.en_US.C	C for AIX PDF documentation — English only
IMNSearch.rte	NetQuestion Search Engine
IMNSearch.rte.DBCS	NetQuestion DBCS search engine
IMNSearch.rte.SBCS	NetQuestion SBCS search engine
IMNSearch.rte.httpdlite	NetQuestion web server
vatools.html.help	VisualAge Tools Help
vatools.msg.LANG.html.help	VisualAge Tools Messages Help

## Filesets Required for C++ Compiler

The following filesets are included in the C++ compiler.

**Note:** In addition to the filesets listed below, all the filesets required for the C compiler are also required for the C++ compiler.

### Filesets Required for Command-Line C++ Compiler

Fileset Name	Fileset Description
vacpp.cmp.aix43.lib	VisualAge C++ Libraries for AIX 4.3
vacpp.cmp.aix43.tools	VisualAge C++ Tools for AIX 4.3
vacpp.cmp.aix50.lib	VisualAge C++ Libraries for AIX 5.1
vacpp.cmp.aix50.tools	VisualAge C++ Tools for AIX 5.1
vacpp.cmp.core	VisualAge C++ Compiler
vacpp.cmp.include	VisualAge C++ Compiler Include Files
vacpp.cmp.lib	VisualAge C++ Libraries
vacpp.cmp.rte	VisualAge C++ Compiler Application Runtime
vacpp.cmp.tools	VisualAge C++ Tools
vacpp.lic	VisualAge C++ for AIX LUM License Files
vacpp.memdbg.aix43.lib	VisualAge C++ User Heap and Memory Debugger Libraries for AIX 4.3
vacpp.memdbg.aix43.rte	VisualAge C++ User Heap and Memory Debugger Runtime for AIX 4.3
vacpp.memdbg.aix50.lib	VisualAge C++ User Heap and Memory Debugger Libraries for AIX 5.1
vacpp.memdbg.aix50.rte	VisualAge C++ User Heap and Memory Debugger Runtime for AIX 5.1
vacpp.memdbg.lib	VisualAge C++ User Heap and Memory Debugger Libraries

### Filesets Required for Command-Line C++ Compiler

Fileset Name	Fileset Description
vacpp.memdbg.rte	VisualAge C++ User Heap and Memory Debugger Runtime
vacpp.msg.LANG	VisualAge C++ Compiler Messages

## Filesets Required for C++ Compiler Online Help

The following filesets are included in the C++ compiler online help.

### Filesets Required for C++ Compiler Online Help

Fileset Name	Fileset Description
vacpp.Dt.common	VisualAge C++ Desktop Integration Common Files
vacpp.Dt.help	VisualAge C++ Help Desktop Integration
vacpp.html.LANG	VisualAge C++ HTML Documentation
vacpp.html.DBCS	VisualAge C++ Documentation Double Byte Common Files
vacpp.html.SBCS	VisualAge C++ Documentation Single Byte Common Files
vacpp.html.common	VisualAge C++ Documentation Common Files
vacpp.html.help	VisualAge C++ HTML Help Engine
vacpp.loc.LANG.Dt.help	VisualAge C++ Help Desktop
vacpp.loc.LANG.cmp	VisualAge Compiler C++ Locale Data
vacpp.pdf.LANG	VisualAge C++ PDF Documentation — English only
vacpp.pdf.common	VisualAge C++ PDF Documentation
IMNSearch.rte	NetQuestion Search Engine
IMNSearch.rte.DBCS	NetQuestion DBCS Search Engine
IMNSearch.rte.SBCS	NetQuestion SBCS Search Engine
IMNSearch.rte.httpdlite	NetQuestion Web Server
vatools.html.help	VisualAge Tools Help
vatools.msg.LANG.html.help	VisualAge Tools Messages Help

## Filesets Required for IBM Distributed Debugger

The following filesets are included in IBM Distributed Debugger.

### Filesets Required for IBM Distributed Debugger

Fileset Name	Fileset Description
idebug.client.extras	Debugger Interpreted Engine for OS/390®
idebug.client.gui	Debugger Graphical User Interface
idebug.client.olt	Object Level Trace Viewer
idebug.engine.compiled	Debugger Engine for Compiled Languages
idebug.engine.interpreted	Debugger Engine for Interpreted Languages
idebug.help.en_US	Debugger help — English only
idebug.msg.LANG.engine	Debugger Engine Messages
idebug.msg.LANG.olt	Object Level Trace Messages

#### Filesets Required for IBM Distributed Debugger

Fileset Name	Fileset Description
idebug.rte.hpj	High-Performance Java™ Runtime
idebug.rte.jre	Java Runtime Environment
idebug.rte.olt.Java	Object Level Trace Java Runtime
idebug.rte.olt.client	Object Level Trace Client Controller
idebug.server.olt	Object Level Trace Server

## Filesets Required to Run LUM and SMIT Client GUIs

You must have the following IPF/X filesets on your computer before you can run the LUM and SMIT client GUIs. They are provided on the product CD for your convenience.

#### Filesets Required to Run LUM and SMIT Client GUIs

Fileset Name	Fileset Description
ipfx.adt	IBM Information Presentation Facility Development Kit/6000
ipfx.msg.LANG.adt	IPF/X development kit message and grammar files
ipfx.msg.LANG.rte	IPF/X runtime message and grammar files
ipfx.rte	IBM Information Presentation Facility/6000

## C++ Runtime Filesets Required to Run VisualAge C++ for AIX

You must have the following C++ runtime filesets on your computer before you can run VisualAge C++ for AIX. They are provided on the product CD for your convenience.

#### C++ Runtime Filesets Required to Run VisualAge C++ for AIX

Fileset Name	Fileset Description
xlC.adt.include	C++ Application Development Toolkit
xlC.aix43.rte	C++ Runtime for AIX 4.3
xlC.aix50.rte	C++ Runtime for AIX 5.1
xlC.msg.LANG	C++ Runtime Messages
xlC.rte	C++ Runtime

## Optional VisualAge C++ for AIX Filesets

The following optional filesets are not required for any VisualAge C++ for AIX component.

#### Optional VisualAge C++ for AIX Filesets

Fileset Name	Fileset Description
vac.ndi	C for AIX Non-Default Install Script
vacpp.ndi	VisualAge C++ Non-Default Install Script
vacpp.samples.ansicl	VisualAge C++ Sample Files

## Removed Filesets

The following filesets were included with VisualAge C++ for AIX, Version 5.0. They are not shipped with VisualAge C++ for AIX, Version 6.0, since they contain functionality which is no longer supported by IBM.

**Note:** To avoid unexpected results, you should uninstall the previous version of the compiler before installing IBM VisualAge C++ Professional for AIX, Version 6.0. At minimum, ensure that the following filesets are uninstalled before installing IBM VisualAge C++ Professional for AIX, Version 6.0.

Filesets Not Included with this Version of VisualAge C++ for AIX

Fileset Name	Fileset Description
vacpp.Dt.dax	VisualAge C++ Data Access Builder Desktop Integration
vacpp.Dt.ide	VisualAge C++ IDE Desktop Integration
vacpp.Dt.ipf	IPF/X Desktop Integration
vacpp.Dt.perf	VisualAge C++ Performance Analyzer Desktop Integration
vacpp.Dt.techide	VisualAge C++ IDE Tech Preview Desktop Integration
vacpp.Dt.vb	VisualAge C++ Visual Builder Desktop Integration
vacpp.cmp.extension	VisualAge C++ Extension Interface
vacpp.cmp.incremental	VisualAge C++ Incremental Compiler
vacpp.dax.adt	VisualAge C++ Data Access Builder Toolkit
vacpp.dax.rte	VisualAge C++ Data Access Builder Runtime
vacpp.ide	VisualAge C++ IDE
vacpp.ioc.aix41.lib	IBM Open Class Library AIX 4.1 and 4.2 Static Libraries
vacpp.ioc.aix41.rte	IBM Open Class Library AIX 4.1 and 4.2 Application Runtime
vacpp.ioc.aix43.lib	IBM Open Class Library AIX 4.3 Static Libraries
vacpp.ioc.aix43.rte	IBM Open Class Library AIX 4.3 Application Runtime
vacpp.ioc.aix50.lib	IBM Open Class Library AIX 5.1 Static Libraries
vacpp.ioc.aix50.rte	IBM Open Class Library AIX 5.1 Application Runtime
vacpp.ioc.include	IBM Open Class Library Include Files
vacpp.ioc.lib	IBM Open Class Library Static Libraries
vacpp.ioc.rte	IBM Open Class Library Application Runtime
vacpp.loc.LANG.Dt.dax	VisualAge C++ Data Access Builder Desktop
vacpp.loc.LANG.Dt.ipf	VisualAge C++ IPFX Desktop
vacpp.loc.LANG.Dt.vb	VisualAge C++ Visual Builder Desktop
vacpp.msg.LANG.dax	Data Access Builder Messages
vacpp.msg.LANG.ide	VisualAge C++ IDE Messages
vacpp.msg.LANG.ioc.rte	IBM Open Class Library Runtime Messages
vacpp.msg.LANG.rescmp	VisualAge C++ Resource Compiler Messages
vacpp.msg.LANG.vb	Visual Builder Messages
vacpp.rescmp	VisualAge C++ Resource Compiler

### Filesets Not Included with this Version of VisualAge C++ for AIX

Fileset Name	Fileset Description
vacpp.samples.dax	Data Access Builder Samples
vacpp.samples.ioc	IBM Open Class Library Samples
vacpp.samples.vb	VisualAge C++ Visual Builder Samples
vacpp.source.ioc	IBM Open Class Library Source
vacpp.tutorial.LANG	VisualAge C++ Tutorial
vacpp.tutorial.common	VisualAge C++ Tutorial — Common Files
vacpp.vb.adt	Visual Builder Toolkit
vacpp.vb.include	Visual Builder Include Files
vacpp.vb.lib	Visual Builder Libraries
vacpp.vb.rte	VisualAge C++ Visual Builder Runtime

---

## Upgrading the AIX Operating System

The minimum operating system requirement for IBM VisualAge C++ Professional for AIX, Version 6.0 is AIX 4.3.3. When you upgrade your operating system (for example, to AIX 5.1), you need to consider the impact on VisualAge C++ for AIX. There are two possible migration paths.

Recommended migration path:

1. Uninstall VisualAge C++ for AIX.
2. Upgrade your operating system.
3. Install VisualAge C++ for AIX.

Brief migration path:

1. Upgrade your operating system.
2. Make sure all filesets support the correct level of operating system. For example, if you upgraded from AIX 4.3.3 to AIX 5.1, and you have a fileset named *this.aix43.fileset*, you must install *this.aix50.fileset*.

**Note:** You must have installed the xlC.rte 5.0.2.1 fileset before you can upgrade your operating system. This fileset contains a packaging change that allows migration installs of AIX 5.1 from an AIX 4.3 system. If you do not install this fileset on your AIX 4.3 operating system, your system will not boot up after you migrate to AIX 5.1. This fix is available on the <http://www.ibm.com/software/ad/vacpp> Web site. Click **Download** and look for "VisualAge C++ for AIX 5.0.2.1 runtime PTF".



---

## Chapter 4. Migrating to VisualAge C++ for AIX, Version 6

This section contains information about the prerequisites, requirements, and environment considerations for migrating from earlier versions of the product. The following questions are answered about migrating to IBM VisualAge C++ Professional for AIX, Version 6.0 are answered:

- What tools should you use for future application development if you were previously using withdrawn tools?
- What are the aspects of existing applications must you change in order to be able to compile the existing application on the new version of the compiler?

### Compiler

For information about migrating an earlier version of VisualAge C++ for AIX to IBM VisualAge C++ Professional for AIX, Version 6.0, see Chapter 3, “Installing VisualAge C++ for AIX” on page 29.

### Source Code

In general, your Version 5.0 compiler source code will compile on IBM VisualAge C++ Professional for AIX, Version 6.0. However, some changes to compiler options, and the support of the C99 standard, may cause some errors if you try to recompile your existing code. For changes to the compiler that may affect your source code, see “Conformance to Industry Standards” on page 11 and “Efficiency, Optimization, and Special Options” on page 7.

### Withdrawn Functionality

If you used to use the tools, utilities and classes listed below, which are no longer supported on VisualAge C++ for AIX, you may need to change your work habits. The withdrawn tools and runtimes are:

- Incremental C++ compiler
- IBM Open Class Library (IOC)
- Visual Builder (VB)
- Data Access Builder (DAX)
- Integrated development environment (IDE), including integrated debugger and LPEX editor
- Performance Analyzer
- Resource Editor (ire)
- Resource Compiler (irc)
- Resource Conversion Utility (irconv)

### RELATED REFERENCES

- “Incremental Compiler Transition” on page 46
- “IBM Open Class Library Transition” on page 47
- “Visual Builder Transition” on page 47
- “Data Access Builder (DAX) Transition” on page 48
- “Data Access Class Library and Generated Code” on page 48
- “Integrated Development Environment (IDE) Transition” on page 49
- “Performance Analyzer Transition” on page 49
- “Resource Editing Tools Transition” on page 49

---

## Incremental Compiler Transition

Incremental compilation is no longer supported by VisualAge C++ for AIX. The command-line compiler that you should use for application development compiles your code in a batch process, one complete translation unit at a time.

If you were using the incremental compiler, you need to convert your existing incremental compilation configuration files into command-line batch compiler makefiles.

Configuration files used by the incremental compiler are similar in concept to makefiles used by most make utilities. The target statement in a configuration file is synonymous with the target in a makefile. Only the syntax is different. The source statement is synonymous with a dependency. To convert these constructs, use the following makefile syntax:

```
target : dependency ; command
```

Target, source, and run directives in the existing configuration file can be mapped to targets, dependencies, and commands in the new makefile.

Commands in configuration files are implied by the type of file specified by the source and target statements.

For example, the following configuration file segment compiles a C++ source file into an object file:

```
target "a.o" {  
    source "a.C"  
}
```

An equivalent makefile segment is:

```
a.o: a.C  
    x1C [options] -c a.C
```

If the target was an executable file, then you would need an additional makefile segment for linking:

```
a.out : a.o  
    x1C [options] a.o
```

The configuration file uses the **run** command, and specifies the order of command execution by the before and after modifiers, such as:

```
run (before|after) command
```

In the makefile, the order of command execution is specified by the order of targets and dependencies. A makefile segment for the example above looks like this:

```
a.o : runfirst a.c  
    x1C [options] ;c a.C  
  
runfirst :  
    command
```

### RELATED REFERENCES

- Chapter 4, “Migrating to VisualAge C++ for AIX, Version 6” on page 45
- “Equivalent Incremental and Batch Compiler Options” on page 51

---

## IBM Open Class Library Transition

The IBM Open Class (IOC) Library is a library of C++ classes provided with IBM VisualAge C++ Professional for AIX, Version 4.0 and IBM VisualAge C++ Professional for AIX, Version 5.0. It was also provided on the obsolete IBM C Set ++<sup>®</sup> for AIX, Version 3 and IBM C and C++ Compilers for OS/2<sup>®</sup>, AIX and for Windows NT, Version 3.6 products.

Since IBM is standardizing on the C++ Standard Library, including the Standard Template Library (STL) and other features of the ISO C++ 1998 Standard, support for the IBM Open Class Library has been removed from these products, as well as all other C++ compilers on different platforms. Read the *IBM Open Class Library Transition Guide* to determine whether your application uses IOC, what version of IOC it is using, and the general migration suggestions for application owners whose applications use the IOC. *IBM Open Class Library Migration Guide* also contains migration suggestions for most classes included in the IBM Open Class Library.

In general, where an overlap in functions exists between the IBM Open Class Library and the C++ Standard Library (including the Standard Template Library), the following is recommended:

- Use the Standard Template Library (STL) containers, iterators, and algorithms instead of the IOC collection classes.
- Use the Standard C++ exception classes instead of the IOC exception classes.
- Use the Standard C++ string template classes instead of the IOC string classes.

However, there are many classes in the IBM Open Class Library for which there is no equivalent in the C++ Standard Library. *IBM Open Class Library Transition Guide* identifies some of the options available to application owners to deal with this situation. The decision as to which option is best depends on the version of the IBM Open Class Library you use and the extent to which you use classes without an equivalent replacement in the C++ Standard Library.

**Note:** Although the UNIX Systems Laboratories (USL) I/O Stream Class Library and complex mathematics classes are not being removed at this time, it is recommended that you migrate to the C++ Standard iostream and complex classes. This is especially important if you are migrating other IOC streaming classes to Standard C++ Library streaming classes, since you cannot combine USL and Standard C++ Library streams in one application.

### RELATED REFERENCES

- Chapter 4, “Migrating to VisualAge C++ for AIX, Version 6” on page 45
- USL I/O Streaming in *VisualAge C++ for AIX Programming Topics and Utilities*
- Complex Mathematics Library Overview in *VisualAge C++ for AIX Programming Topics and Utilities*

---

## Visual Builder Transition

The Visual Builder is a visual composition editor for assembling applications visually from IBM Open Class components. This tool is no longer provided with the compiler. Existing applications may be affected, and will need to be migrated. If your existing applications use IBM Open Class components, see “IBM Open Class Library Transition”.

#### RELATED REFERENCES

- Chapter 4, “Migrating to VisualAge C++ for AIX, Version 6” on page 45

---

## Data Access Builder (DAX) Transition

Data Access Builder is an application development tool used to create data access classes customized for existing relational database tables. This tool was invoked as part of a project from the IBM VisualAge C++ Integrated Development Environment, and is no longer provided with the compiler. It was used to generate C++ source code (classes) to access data. These Data Access classes could also be used directly in your C++ programs to provide database access.

If your existing applications contains the Data Access classes listed below, you will no be able to recompile these applications using IBM VisualAge C++ Professional for AIX, Version 6.0. You will need to migrate these existing applications.

For additional information about integrating IBM DB2 Universal Database® data into your application, see your DB2® programming and data access documentation.

## Data Access Class Library and Generated Code

The Data Access classes enabled users to connect to databases, map database tables to objects, and work with the data inside the databases by using object-oriented programming. The base classes in the Data Access Class Library provided generic methods for accessing and manipulating database information, interacting with other builders, handling messages and exceptions, and working with large objects. The generated classes extended the base classes and implemented operations tailored for specific situations. In most cases, developers used the generated classes in their data access applications.

Use the lists below to determine whether your application uses any Data Access classes. If it does, you will need to migrate these classes.

The Data Access Class Library included the following classes.

- IDatastoreBase
- IDatastoreBaseCLI
- IDatastoreODBC
- IDatastoreDB2
- IDatastore
- IPersistentObject
- IPODataId
- IDAManager
- IDAException
- IDALobDO
- IDAClobDO
- IDADBClobDO
- IDABlobDO

Data Access Builder generated the following classes, where *<class>* refers to the application-specific data. For example, *<class>*Datastore could be EmployeeDatastore or StudentDatastore.

- (<class>)Datastore
- (<class>)
- (<class>)DataId
- (<class>)ManagerBase
- (<class>)ManagerTemplate
- (<class>)Manager
- (<class>)DataIdManagerBase
- (<class>)DataIdManagerTemplate
- (<class>)DataIdManager

#### RELATED REFERENCES

- Chapter 4, “Migrating to VisualAge C++ for AIX, Version 6” on page 45

---

## Integrated Development Environment (IDE) Transition

The Integrated Development Environment (IDE) is a visual environment for developing, compiling, and debugging your applications. This tool is no longer shipped with VisualAge C++ for AIX.

There is no impact on existing code.

The Live Parsing Extensible Editor (LPEX) was part of the IDE. Since it is no longer shipped with VisualAge C++ for AIX, you should use your favorite text editor for writing code. For example, you can use the INed Editor and vi editor that are shipped with your AIX operating system.

The Integrated Debugger was part of the IDE. Since it is no longer shipped with VisualAge C++ for AIX, we recommend you use the standalone Distributed Debugger. For more information, see “IBM Distributed Debugger” on page 2. You can also use the **dd** debug tool shipped with the AIX operating system.

#### RELATED REFERENCES

- Chapter 4, “Migrating to VisualAge C++ for AIX, Version 6” on page 45

---

## Performance Analyzer Transition

The Performance Analyzer tool was used to examine how your program used system resources. As a part of the IDE, it is no longer provided with the compiler. There is no migration impact on existing code.

#### RELATED REFERENCES

- Chapter 4, “Migrating to VisualAge C++ for AIX, Version 6” on page 45

---

## Resource Editing Tools Transition

The following tools that were used to edit, compile, and convert application resources such as message strings, pointers, and menus are no longer shipped with VisualAge C++ for AIX.

- Resource Editor (ire)
- Resource Compiler (irc)

- Resource Conversion Utility (irconv)

There is no migration impact on existing code.

**RELATED REFERENCES**

- Chapter 4, “Migrating to VisualAge C++ for AIX, Version 6” on page 45

## Appendix. Equivalent Incremental and Batch Compiler Options

VisualAge C++ for AIX, Version 5.0 included both an incremental C++ compiler, with a C compiler extension, and batch C and C++ compilers. For both C and C++, you could control the compile and link stages of a build by specifying options in an incremental configuration file (.icc). Now that the incremental compiler has been removed, you can specify these options in your source file, a traditional makefile configuration file, or on the command line when invoking the compiler. Use the following table to determine equivalent incremental and batch compiler options.

Refer to the *VisualAge C++ for AIX Compiler Reference* for information about these options.

The syntax of configuration file options indicates whether the option relates to code generation, optimization, the C or C++ language, the link stage, or other features of builds.

The table below lists the compiler options equivalent to incremental configuration file options.

For more information about linking options beginning with -b or -Wl, see the `ld` command documentation provided with your AIX operating system.

In the option syntax, the following rules apply:

- The characters [ and ] enclose optional items.
- The characters ( and ) enclose a list of alternatives.
- The character | indicates alternatives.
- The character \* denotes zero or more items.

For example `-qkeyword=string[:string]*` is equivalent to `-qkeyword=string`, `-qkeyword=string1:string2`, `-qkeyword=string1:string2:string3` and so on. Another example: `-qtbtable=(none|small|full)` is equivalent to `-qtbtable=none`, or `-qtbtable=small`, or `-qtbtable=full`.

Equivalent incremental and traditional compiler options

Incremental Option	Traditional Compiler Option	Option Description
<code>alloc(debug[,yes])</code> <code>alloc(debug,no)</code>	<code>-qheapdebug</code> <code>-qnoheapdebug</code>	Enables debug versions of memory management functions.
<code>debug(maxErrors, number)</code>	<code>-qmaxerr=number[:severity_level]</code>	Instructs the compiler to halt compilation when a specified number of errors is reached.
<code>define(name[, def])</code>	<code>-Dname[=[def]]</code>	Define preprocessor macro <i>name</i> as in #define directive. If <i>def</i> is not specified, 1 is assumed.
<code>file(genAsm[,yes])</code> <code>file(genAsm, no)</code>	<code>-S</code>	Generate an assembler language (.s) file
<code>file(genObject[,yes])</code> <code>file(genObject,name)</code> <code>file(genObject,no)</code>	<code>-oname</code>	Name generated executable or object file.

## Equivalent incremental and traditional compiler options

Incremental Option	Traditional Compiler Option	Option Description
file(genProto[, yes]) file(genProto,no) file(genProto, withParameterNames)	-qgenproto -qnoqgenproto -qgenproto=parmnames	Generate ANSI prototypes from K&R function definitions (with or without the names of parameters included).
file(makeDep[,yes]) file(makeDep,no)	-qmakedep -qnomakedep	Creates an output file that contains targets suitable for inclusion in a description file for the AIX make command.
file(makeDep[,yes])	-M	Generate information to be included in a "make" description file; output goes to .u file.
file(once[, yes]) file(once, no)	This function is no longer supported by VisualAge C++ for AIX.	Process #include files only once.
file(syntaxOnly[,yes]) file(syntaxOnly,no)	-qsyntaxonly -qnosyntaxonly	Causes the compiler to perform syntax checking without generating an object file.
file(tabSize, <i>number</i> )	-qtabsize= <i>size</i>	Change the length of tabs in your source file.
gen(align, <i>name</i> )	-qalign= <i>name</i>	Specify alignment of data items.
gen(arch, <i>name</i> )	-qarch= <i>name</i>	Specifies the architecture on which the executable program will be run.
gen(check, zeroDivide[, yes]) gen(check,zeroDivide, no) gen(check, nullPointer[,yes]) gen(check,nullPointer,no) gen(check, bounds[,yes]) gen(check,bounds,no)	-qcheck -qnocheck -qcheck= <i>suboptions</i>	Generate code that performs various types of runtime checking.
gen(debugUnreferenced[,yes]) gen(debugUnreferenced,no) link(staticSymbols) link(staticSymbols,no)	-qsymtab=unref -qsymtab=static	Set certain back end options.
gen(debugunreferenced[,yes]) gen(debugunreferenced,no)	-qdbxextra -qnodbxextra	Produce a symbol table for unreferenced variables.
gen(eh[, yes]) gen(eh, no)	-qeh -qnoeh	Controls whether C++ exception handling is enabled in the module being built.
gen(enumSize, <i>size</i> )	-qenum= <i>size</i>	Specify the amount of storage occupied by enumerations.
gen(externStaticLinkage[, yes]) gen(externStaticLinkage,no)	-qxcall -qnoxcall	Generate code to static routines within a compilation unit as if they were external routines.
gen(fdpr, yes)	-qfdpr -qnofdpr	Collect information about programs for use with the AIX <b>fdpr</b> performance-tuning utility.
gen(float, ...) gen(float, <i>string</i> [,yes]) gen(float, <i>string</i> ,no)	-qfloat= <i>opt[:opt]*</i>	Specifies various floating-point options to speed up or improve the accuracy of floating-point calculations.
gen(float, fold[, yes]) gen(float,fold, no)	-qfold -qnofold	Specifies that constant floating-point expressions are to be evaluated at compile time.
gen(floatTrap, <i>option</i> [,yes]) gen(floatTrap, <i>option</i> ,no)	-qflttrap= <i>opt[:opt]*</i> -qnoflttrap	Generate calls to detect and trap floating point exceptions.
gen(funcSect[, yes]) gen(funcSect,no)	-qfuncsect -qnofuncsect	Place instructions for each function in a separate object file, control section or csect.
gen(initAuto[, yes]) gen(initAuto, <i>number</i> ) gen(initAuto, no)	-qinitauto= <i>number</i>	Initialize automatic storage to <i>hh</i> , where <i>hh</i> is a hexadecimal value.

## Equivalent incremental and traditional compiler options

Incremental Option	Traditional Compiler Option	Option Description
gen(inlinePointerGlue[, yes]) gen(inlinePointerGlue,no)	-qinlgue -qnoinlgue	Generate fast external linkage by inlining the code (pointer glue code) necessary for calls via a function pointer and calls to external procedures.
gen(libansi) gen(libansi,no)	-qlibansi -qnolibansi	Process functions with names that match ANSI C library names as being the ANSI C functions.
gen(lineOnlyDebug[,yes]) gen(lineOnlyDebug,no)	-qlinedebug -qnolinedebug	Generates abbreviated line number and source filename information for the debugger.
gen(longDouble[, yes]) gen(longDouble,no)	-qldbl128 -qnoldbl128 -qlongdouble -qnolongdouble	Represent long doubles as 128 bit or 64 bit values.
gen(objectModel, <i>model</i> )	-qobjmodel= <i>model</i>	Select the default C++ object model for the compilation unit.
gen(profile, <i>bsd</i> )	-pg	Generate profiling support code including BSD profiling support. Enables profiling with <b>gprof</b> .
gen(profile, <i>ibm</i> )	This function is no longer supported by VisualAge C++ for AIX.	Enables profiling with the Performance Analyzer tool.
gen(profile[, yes])	-p	Enable code for performance analysis. Enables profiling with <b>prof</b> .
gen(readonly[, yes]) gen(readonly,no)	-qro -qnoro	Put string literals in read only area.
gen(readonlyconst) gen(readonlyconst,no)	-qroconst -qnoroconst	Put constant values in read only area.
gen(roundConstFp, <i>mode</i> )	-ymode	Specify compile-time rounding of constant floating-point expressions.
gen(rtti[, yes]) gen(rtti, all) gen(rtti,typeInfo) gen(rtti, dynamicCast) gen(rtti,no)	-qrtti -qnortti	Generate run-time type identification (RTTI) information for the typeid operator and the dynamic_cast operator.
gen(strictinduction[, yes]) gen(strictinduction,no)	-qstrict_induction -qnostrict_induction	Disable loop induction variable optimizations that have the potential to alter the semantics of the program.
gen(vft[, yes]) gen(vft,no)	-qvftable -qnovftable	Determine whether the virtual function table is included in the module that is the target of the build.
incl(externC, <i>prefix</i> ) incl(noExternC, <i>prefix</i> )	-qcinc= <i>prefix</i> -qnocinc= <i>prefix</i>	Include files from the subdirectory <i>prefix</i> and inserts extern "C" { at the beginning of the file and } at the end.
incl(searchpath, <i>path</i> )	-Idir	Search in directory <i>dir</i> for include files that do not start with an absolute path.
incl(searchpathfirst[, yes]) incl(searchpathfirst,no)	-qdirfirst -qnoidirfirst	Specify search order for files.
incl(standardInclude[,yes]) incl(standardInclude,no)	-qstdinc -qnostdinc	If -qnostdinc is specified, the /usr/vacpp/include and /usr/include directories are not searched for include files.
info( <i>class</i> )	-qinfo= <i>class</i>	Produce additional lint-like messages based on <i>class</i> .
lang(allowDollarInNames[, yes]) lang(allowDollarInNames,no)	-qdollar -qnodollar	Allow the dollar sign (\$) in identifier names.

## Equivalent incremental and traditional compiler options

Incremental Option	Traditional Compiler Option	Option Description
lang(anonymousStructs[, yes]) lang(extendedAnonymousUnions[,yes]) lang(ansiForStatementScopes[, yes]) lang(illformedPointerToMember[,yes]) lang(implicitInt[, yes]) lang(newThrowsException[,yes]) lang(offsetOfNonPODClasses[,yes]) lang(oldDigraphs[, yes]) lang(compatFriendDeclarations[,yes]) lang(compatMath[, yes]) lang(compatTempAccessChecking[,yes]) gen(compatNestedTemplateAlignmentRule) lang(compatTemplateSpecialization[,yes]) lang(trailingEnumCommas[, yes]) lang(allowTypedefAsClassName[,yes]) lang(universalCharacterNames[, yes]) lang(zeroExternArrays[,yes])	-qlanglvl =[no]anonstruct =[no]anonunion =[no]ansifor =[no]illptom =[no]implicitint =[no]newexcp =[no]offsetnonprod =[no]olddigraphs =[no]oldfriend =[no]oldmath =[no]oldtplacc =[no]oldtplalign =[no]oldtplspec =[no]trailenum =[no]typedefclass =[no]jucs =[no]zeroextarray	Set the language level option.
lang(checkNonProto[,yes]) lang(checkNonProto,no)	-qproto -qnoproto	Assert that procedure call points agree with their declarations even if the procedure has not been prototyped.
lang(dbc[,yes]) lang(dbc,no)	-qdbc -qnodbc -qmbcs -qnombcs	Allow use of DBCS. Use the -qmbcs option if your program contains multibyte characters.
lang(digraphs[, yes]) lang(digraphs,no)	-qdigraph -qnodigraph	Permit ANSI digraph and keyword operators.
lang(keyword, <i>string</i> )	-qkeyword= <i>string</i> [: <i>string</i> ]* -qnokeyword= <i>string</i> [: <i>string</i> ]*	Controls whether the specified <i>string</i> is treated as a keyword or an identifier whenever it appears in your C++ source.
lang(level, <i>level</i> )	-qlanglvl= <i>level</i>	Specify language level to be used during compilation.
lang(longlong) lang(longlong,no)	-qlonglong -qnolonglong	Allows long long types in your program.
lang(macPStr[, yes]) lang(macPStr, no)	-qmacpstr -qnomacpstr	Converts Pascal string literals into null-terminated strings where the first byte contains the length of the string.
lang(pascal[, yes]) lang(pascal, no)	-qpascal -nopascal	Accept the keyword 'pascal' as a type modifier.
lang(preserveUnsignedPromotion[, yes]) lang(preserveUnsignedPromotion,no)	-qupconv -qnoupconv	Preserves the unsigned specification when performing integral promotions.
lang(signedBitFields[,yes]) lang(signedBitFields,no)	-qbitfields=signed -qbitfields=unsigned	Specify whether bit fields will be signed or unsigned.
lang(signedChars[, yes]) lang(signedChars,no)	-qchars=signed -qchars=unsigned	Treat plain <i>char</i> variables as signed or unsigned.
lang(slashSlashComment[,yes]) lang(slashSlashComment,no)	-qplusplus -qnoplusplusplus	Permit <code>/// comment</code> that lasts until the end of the current source line, as in C++.
lang(staticInlineLinkage[, yes]) lang(staticInlineLinkage,no)	-qstaticinline -qnostaticinline	Controls whether inline functions are treated as static or extern.
lang(templateDefImpls, parseWithWarnings) lang(templateDefImpls, parseWithErrors) lang(templateDefImpls, dontParse)	-qtmplparse=(warn error no)	Controls whether parsing and semantic checking are applied to template definition implementations or only to template instantiations
link(alignAddr, <i>number</i> )	-Wl,-H <i>number</i>	Set the address alignment for all loadable segments so that the start of each segment is aligned at a multiple of the specified number of bytes.

## Equivalent incremental and traditional compiler options

Incremental Option	Traditional Compiler Option	Option Description
link(alignPage, yes)	-Wl,-K -Wl,-z	Aligns the header, text, data, and loader sections of the output file so that each section begins on a page boundary.
link(bigTOC, yes) link(bigTOC, no)	-bbigtoc -bnobigtoc	Generates extra code to allow the table of contents (TOC) to grow to a size greater than 64KB.
link(dataImported, none) link(dataImported,all) link(dataImported, names) link(dataImportedNames, name)	-qdataimported -qdataimported=name[:name]*	Specify which data items are imported. If no names are specified, all data items are assumed to be imported. This is the default.
link(dataLocal, none) link(dataLocal,all) link(dataLocal, names) link(dataLocalNames, name)	-qdatalocal -qdatalocal=name[:name]*	Specify which data items are local. If no names are specified, all data items are assumed to be local.
link(dataStart, number)	-Wl,-Dnumber,-bPD:number	Set the starting address for the data section of the output file to <i>number</i> .
link(debugFullPath[, yes]) link(debugFullPath,no)	-qfullpath -qnofullpath	Specify path information.
link(debug[, yes]) link(debug, no)	-g	Include traditional debug information in the target.
link(dpcl[, yes]) link(dpcl,no)	-qdpcl -qnodpcl	Generates symbols that tools based on the DynamicProbe Class Library (DPCL) can use to see the structure of an executable file.
link(entry, string) link(entry,no)	-e name -bnoentry	Specify the entry name for the shared object. Used only with <b>mkshrobj</b> command.
link(force)	-r	Permits the output file to be produced even if it contains unresolved symbols.
link(heap, number)	-bmaxdata:number	Set the size of the heap in bytes.
link(libPathOut, string)	-blibpath:string	Override search path used at run time.
link(libSearchPath,string)	-Lstring	Specify search path for library files.
link(libsearchpathprefix, string)	-Zstring	Prefix the names of the library search paths with the <i>string</i> .
link(nameMangling, ansi) link(nameMangling,compat)	-qnamemangling=option	Chooses the name mangling scheme for external symbol names generated from C++ source code.
link(procImported,<none   all   names>) link(procImportedNames,string)	-qprocimported -qprocimported=name[:name]*	Specify which functions are imported.
link(procLocal,<none   all   names>) link(procLocalNames, string)	-qprocllocal -qprocllocal=name[:name]*	Specify which functions are local.
link(procUnknown,<none   all   names>) link(procUnknownNames, string)	-qprocunknown -qprocunknown=name[:name]*	Specify which functions are unknown to be local or imported.
link(runtimeLinking)	-brtl	Enables runtime linking.
link(sharedLibPriority, number)	-qpriority=num	Specifies the priority level for initialization of static constructors.
link(stack,number)	-Wl, -Snumber	Set the total size of the program stack in bytes.
link(static, yes) link(static, no)	-bstatic -bdynamic	Controls how shared objects are processed by the linkage editor.

## Equivalent incremental and traditional compiler options

Incremental Option	Traditional Compiler Option	Option Description
link(staticSymbols) link(staticSymbols,no)	-qstatsym -qnostatsym	Adds user-defined, non-external names that have a persistent storage class, such as initialized and uninitialized static variables, to the name list.
link(strip[, yes]) link(strip,no)	-s	Strip the symbol table.
link(textStart, number)	-W1, -Tnumber	Sets the start address of the text section of the output file to <i>number</i> .
link(traceBackTable,<none   full   small>)	-qtbtable=(non   small   full)	Generate a trace-back table for each function and place it in the text segment at the end of the function code.
link(typeCheck[, yes]) link(typeCheck,no)	-qextchk -qnoextchk	Perform external name type-checking and function call checking.
link(uniqueNames[, yes]) link(uniqueNames, no)	-qunique -qnounique	Generates unique names for static constructor or destructor compilation units.
list(attr[,yes]) list(attr, full)	-qattr -qattr=full	Produces a compiler listing that includes an attribute listing for all identifiers.
list(expAllInc[,yes]) list(expAllInc,no)	-qshowinc -qnoshowinc	Include the source for all included files in the source listing, if the -qsource option is in effect.
list(fullXRef[, yes]) list(fullXRef,no)	-qxref=full	Produce a cross-reference listing containing all names, whether referenced or not; output goes to .lst file.
list(incSource[,yes]) list(incSource,no)	-qsource -qnosource	Produce a source listing; output goes to .lst file.
list(listing, no)	-qnoprint	Direct listing to /dev/null.
list(listing[,yes]) list(listing,no)	-qlist -qnolist	Produce a compiler listing that includes an object listing.
list(minXRef[, yes]) list(minXRef, no)	-qxref -noxref	Produce a cross-reference listing; output goes to .lst file.
list(options[, yes]) list(options, no)	-qlistopt -qnolistopt	Display the settings of all options; output goes to .lst file.
list(reportLevel, sev1) report(level,sev2)	-qflag=sev1[:sev2]	Specify severity level of diagnostics to be reported in source listing <i>sev1</i> , and stderr <i>sev2</i> .
opt(alias, ...)	-qansialias -qnoansialias	Specify which aliasing rules can be used during optimization.
opt(alias, option) opt(aliasAssert,typ[,yes]) opt(aliasAssert,typ,no) opt(aliasAssert,allp[,yes]) opt(aliasAssert,allp,no) opt(aliasAssert,addr[,yes]) opt(aliasAssert,addr,no)	-qassert=option -qassert=(typ   allp   addr)	Specifies the aliasing assertion to be applied to your compilation unit.
opt(alias,...)	-qalias=(typ   allp   addr   ansi)	Specifies the aliasing assertion to be applied to your compilation unit.
opt(autoInline[, yes]) opt(autoInline,no) opt(inline[, yes]) opt(inline,no)	-qinline -qnoinline	Determine whether functions in your code are candidates for inlining.
opt(ignErrno) opt(ignErrno,no)	-qignerrno -qnoignerrno	Tell the optimizer that the program will never refer to or set errno.

## Equivalent incremental and traditional compiler options

Incremental Option	Traditional Compiler Option	Option Description
opt(inline) opt(autoInline) opt(autoInline, <i>lc</i> )	-Q= <i>lc</i> -qinline= <i>lc</i>	Inline if number of source statements in function is less than the number specified in <i>lc</i> .
opt(inline, no) opt(autoInline, no)	-Q! -qnoinline	Do not inline any function.
opt(inline[, yes]) opt(autoInline[,yes])	-Q -qinline	Consider any function for inlining. The default is to inline only those functions declared inline.
opt(inline[, yes]) opt(noInlineFunc, <i>name</i> )	-Q- <i>name[:name]*</i> -qinline- <i>name[:name]*</i>	Do not inline function listed by names.
opt(inline[, yes]) opt(inlineFunc, <i>name</i> )	-Q+ <i>name[:name]*</i> -qinline+ <i>name[:name]*</i>	Attempt to inline functions listed by names.
opt(isolatedCall,...)	-qisolated_call= <i>name[:name]*</i>	Specify that the calls to the function <i>name</i> listed have no side effects.
opt(level, <i>number</i> )	-O -O2 -O3 -O4 -O5	Optimize code.
opt(loopUnroll[, yes]) opt(loopUnroll, <i>number</i> ) opt(loopUnroll, no)	-qunroll -qunroll= <i>number</i> -qnounroll	Allow the optimizer to unroll loops.
opt(maxMem, <i>number</i> )	-qmaxmem= <i>num</i>	Limit the amount of memory used by space intensive optimization to <i>num</i> .
opt(pragmaDisjoint[, yes]) opt(pragmaDisjoint,no) opt(pragmaIsolatedCall[, yes]) opt(pragmaIsolatedCall,no)	-qignprag=disjoint -qignprag=isolated -qignprag=all	Honor or ignore references to #pragma disjoint or isolated_call.
opt(profileDirectedFeedback[, yes])	-qpdf1 -qpdf2	Perform aggressive optimizations with profile directed feedback.
opt(registerSpillSize, <i>number</i> )	-qspill= <i>number</i>	Specify the size of the register allocation spill area.
opt(size[, yes]) opt(size,no)	-qcompact -qnocompact	Optimize code for size.
opt(strict) opt(strict, no)	-qstrict -qnostrict	This option turns off aggressive optimizations which have the potential to alter the semantics of a user's program.
opt(tune, <i>name</i> ) opt(tune,...)	-qtune= <i>name</i>	Specifies the system architecture for which the executable program is optimized.
pp(preprocessOnly) pp(preserveComments)	-C	Preserve comments in preprocessed output.
pp(preprocessOnly) pp(stdout)	-E	Preprocess but do not compile. Output goes to stdout.
pp(preprocessOnly[,yes])	-P	Preprocess but do not compile. Output goes to .i file.
report(level, ...)	-w	Suppress information, language-level, and warning messages.
report(srcMsg[, yes]) report(srcMsg,no)	-qsrcmsg -qnosrcmsg	Reconstruct source lines in error along with the diagnostic messages.
report(treatAsError, <i>message</i> )	-qhaltonmsg= <i>message[:message]*</i>	Treats the specified message like an error message (preventing the creation of the target).
source type(imp) " <i>filename</i> " source " <i>filename</i> .imp"	-bI: <i>filename</i>	Names a file containing a list of imported symbols.

### Equivalent incremental and traditional compiler options

Incremental Option	Traditional Compiler Option	Option Description
undefine( <i>name</i> )	-U <i>name</i>	Undefines a specified identifier defined by the compiler or by the -D option.

---

## Notices

Note to U.S. Government Users Restricted Rights -- use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

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  
B3/KB7/8200/MKM  
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.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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.

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:

## Programming Interface Information

Programming interface information is intended to help you create application software using this program.

General-use programming interface 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 are trademarks of the International Business Machines Corporation in the United States, or other countries, or both:

- AIX
- AIX 5L
- IBM
- Open Class
- OS/390
- PowerPC
- VisualAge

Java and Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries or both.

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

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

---

## Industry Standards

The following standards are supported:

- The C language is consistent with the International Standard for Information Systems-Programming Language C (ISO/IEC 9899-1999 (E)).
- The C++ language is consistent with the International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:1998).