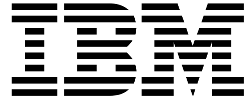
VisualAge® C++ Professional for AIX®



Preprocessing Directives - #pragma

Version 5.0

Note!

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

Edition Notice

This edition applies to Version 5.0 of IBM VisualAge C++ and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

© Copyright International Business Machines Corporation 1998, 2000. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices v
Programming Interface Information vii
Trademarks and Service Marks vii
Industry Standards viii

About This Book. ix

Chapter 1. Preprocessor Directives - #pragma 1

Chapter 2. Overview of the #pragma Options 5
#pragma align (AIX) 5
#pragma alloca (C, AIX) 5
#pragma chars 6
#pragma comment 6
#pragma define (C++) 7
#pragma disjoint 8
#pragma entry 9
#pragma enum 9
#pragma export 10
#pragma hdrfile (C). 11
Examples using #pragma hdrfile (C) 12
#pragma hdrstop (C) 13
Examples using #pragma hdrstop (C). 13
#pragma import 14
#pragma info 15
Example: #pragma isolated_call. 15
#pragma langlvl (C) 16
#pragma leaves 17
#pragma map 18
#pragma namemangling (C++) 19
#pragma object_model (C++) 19
#pragma options (AIX) 20
#pragma priority (C++) 22
#pragma reachable 23
#pragma report (C++) 24
#pragma strings 26

Contacting IBM® 29

© Copyright IBM Corp. 1998, 2000

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.

© Copyright IBM Corp. 1998, 2000

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.
1150 Eglinton Avenue East
Toronto, Ontario M3C 1H7
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.

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:

® (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. ® Copyright IBM Corp. 1998, 2000. All rights reserved.

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.

Warning: 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
AS/400
DB2
CICS
C Set ++
IBM
Network Station
Object Connection
OS/2
OS/390
OS/400
Open Class
Operating System/2
Operating System/400
PowerPC 403
PowerPC 601
PowerPC 603
PowerPC 604
Presentation Manager
RS/6000
S/390
SAA
Systems Application Architechture
TeamConnection
VisualAge
WebSphere
Workplace Shell

Lotus, Lotus Notes, and Domino are trademarks or registered trademarks of the Lotus Development Corporation in the United States, or other countries, or both.

Tivoli Management Environment, TME 10, and Tivoli Module Designer are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

Encina and DCE Encina Lightweight Client are trademarks of Transarc Corporation in the United States, or other countries, or both.

Microsoft, Win32, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the U.S. and other countries licensed exclusively through X/Open Company Limited.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, or other countries, or both.

C-bus is a registered trademark of Corollary, Inc.

PC Direct is a registered tradmark of Ziff Communicatoins Company and is used by IBM Corporation under license

Other company, product, and service names, which may be denoted by a double asterisk(**), may be trademarks or service marks of others.

Industry Standards

VisualAge C++ Professional for AIX, Version 5.0 supports the following standards:

- The C language is consistent with the International Standard C (ANSI/ISO-IEC 9899–1990 [1992]). This standard has officially replaced American National standard for Information Systems-Programming Language C (X3.159–1989) and is technically equivalent to the ANSI C standard. VisualAge C++ supports the changes adopted into the C Standard by ISO/IEC 9899:1990/Amendment 1:1994.
- The IBM Systems Application Architecture (SAA) C Level 2 language definition.
- The C++ language is consistent with the International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:1998).
- The ISO/IEC 9945–1:1990/IEEE POSIX 1003.-1990 standard.
- The X/Open Common Applications Environment Specifications, System Interfaces and Headers, Issue 4.

About This Book

The information in this PDF document is also available in the online help.

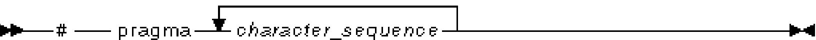
To find this information, or any topics listed in this document as Related Concepts, Related Tasks, or Related References, simply type the topic title into the search bar in the top frame of your browser in the online help.

For some topics, the suggested references may already be contained in this document. In such cases, there is a cross-reference to the page on which the related topic appears.

© Copyright IBM Corp. 1998, 2000

Chapter 1. Preprocessor Directives - #pragma

A pragma is an implementation-defined instruction to the compiler. It has the general form given below, where *character_sequence* is a series of characters that giving a specific compiler instruction and arguments, if any.



The *character_sequence* on a pragma is subject to macro substitutions, unless otherwise stated. More than one pragma construct can be specified on a single #pragma directive. The compiler ignores unrecognized pragmas, issuing an informational message indicating this.

The following pragmas are available:

Platform and Language Restrictions	#pragma	Description
AIX	"#pragma align (AIX)" on page 5	Aligns data items within structures.
C AIX	"#pragma alloca (C, AIX)" on page 5	Provides an inline version of the function alloca(size_t size).
OS/2 WIN	alloc_text	Groups functions into separate 32-bit code segments.
	"#pragma chars" on page 6	Sets the sign type of character data.
OS/2 WIN	code_seg	Marks the current code section in effect.
	"#pragma comment" on page 6	Places a comment into the object file.
OS/2 WIN	data_seg	Places static and external variables in different 32-bit data segments.
C++	"#pragma define (C++)" on page 7	Forces the definition of a template class without actually defining an object of the class.
	"#pragma disjoint" on page 8	Lists the identifiers that are not aliased to each other within the scope of their use.
(C++ AIX) & OS/2 WIN	"#pragma entry" on page 9	Specifies the function to be used as the entry point for the application being built.
	"#pragma enum" on page 9	Specifies the size of enum variables that follow.

© Copyright IBM Corp. 1998, 2000

Platform and Language Restrictions	#pragma	Description
(C++ AIX) & OS/2 WIN	"#pragma export" on page 10	Declares that a DLL function is to be exported and specifies the name of the function outside the DLL.
OS/2 WIN	handler	Registers an exception handler for a function.
C++	hashome	Informs the compiler that the specified class has a home module that will be specified by the IsHome pragma.
C	"#pragma hdrfile (C)" on page 11	Specifies the filename of the precompiled header to be generated and/or used.
C	"#pragma hdrstop (C)" on page 13	Manually terminates the initial sequence of #include directives being considered for precompilation.
C++	implementation	Tells the compiler the name of the file containing the function-template definitions that correspond to the template declarations in the include file which contains the pragma.
(C++ AIX) & OS/2 WIN	"#pragma import" on page 14	Lets you import a function or a variable from a DLL using either an ordinal number or a name different from the one that it has in the DLL.
	"#pragma info" on page 15	Controls the diagnostic messages generated by the info(...) compiler options.
C++	ishome	Informs the compiler that the specified class's home module is the current compilation unit.
	isolated_call	Lists functions that do not alter data objects visible at the time of the function call.
C	"#pragma langlvl (C)" on page 16	Selects the C language level for compilation.
	"#pragma leaves" on page 17	Takes a function name and specifies that the function never returns to the instruction after the function call.
OS/2 WIN	library	This tells the linker to pull in the appropriate libraries at link time.

Platform and Language Restrictions	#pragma	Description
C OS/2 WIN	linkage	Identifies the linkage or calling convention used on a function call.
	"#pragma map" on page 18	Tells the compiler that all references to an identifier are to be converted to a new name.
C OS/2 WIN	margins	Specifies the columns in the input line that are to be scanned for input to the compiler.
AIX	mc_func	Specifies machine instructions for a particular function.
	"#pragma namemangling (C++)" on page 19	Sets the maximum length for external names generated from source code.
C++	"#pragma object_model (C++)" on page 19	Specifies the object model to use for the structures, unions, and classes that follow it.
AIX	"#pragma options (AIX)" on page 20	Specifies options to the compiler in your source program.
	option_override	Specifies alternate optimization options for specific functions.
OS/2 WIN	pack	Specifies the alignment rules to use for the structures, unions, and classes that follow it.
C OS/2 WIN	page	Skips the number of pages of the generated source listing. The number of pages is defined by the user.
C OS/2 WIN	pagesize	Sets the number of lines per page to user-defined value for the generated source listing.
C++	"#pragma priority (C++)" on page 22	Specifies the order in which static objects are to be initialized at run time.
	"#pragma reachable" on page 23	Declares that the point after the call to a routine marked reachable can be the target of a branch from some unknown location.

Platform and Language Restrictions	#pragma	Description
AIX	reg_killed_by	Specifies those registers which value will be corrupted by the specified function. It must be used together with #pragma mc_func.
C++	"#pragma report (C++)" on page 24	Controls the generation of specific messages.
C OS/2	seg16	Shares a data object between 16-bit and 32-bit processes.
C OS/2 WIN	sequence	Defines the section of the input line that is to contain sequence numbers.
C OS/2 WIN	skip	Skips the specified number of lines of the generated source listing.
C OS/2	stack16	Specifies the size of the stack to be allocated for calls to 16-bit routines.
OS/2 WIN	stack_align	Applies the stack alignment rules to the user defined functions.
	"#pragma strings" on page 26	Sets storage type for strings.
C OS/2 WIN	subtitle	Places the text specified by subbtitle on all subsequent pages of the generated source listing.
C OS/2 WIN	title	Places the text specified by the user on all subsequent pages of the generated source listing.
OS/2 WIN	weak	Adds an alternate function name with weak binding for the specified function.

RELATED CONCEPTS

Preprocessor Directives - An Overview

RELATED REFERENCES

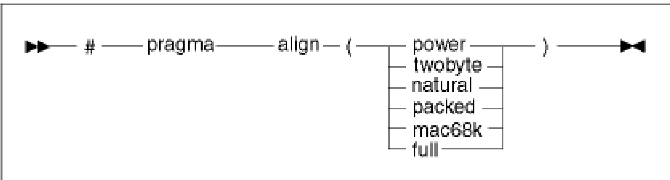
#pragma Preprocessor Directives for Parallel Processing (C Only)

Chapter 2. Overview of the #pragma Options

The following is a summary of each #pragma option.

#pragma align (AIX)

The *#pragma align* directive specifies that the compiler should align data items within structures.



OS/2 WIN On the Intel platform, use *#pragma pack* instead.

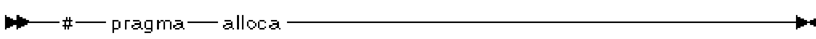
RELATED CONCEPTS
Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

#pragma alloca (C, AIX)

Description

The *#pragma alloca* directive specifies that the compiler should provide an inline version of the function *alloca(size_t <size>)*. The function alloca(size_t <*size*>) can be used to allocate space for an object. The amount of space allocated is determined by the value of <*size*>, which is measured in bytes. The allocated space is put on the stack.

Syntax



You must include the #pragma alloca directive to have the compiler provide an inline version of alloca.

Once specified, it applies to the rest of the file and cannot be turned off. If a source file contains any functions that you want compiled without #pragma alloca, place these functions in a different file.

RELATED CONCEPTS
Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

© Copyright IBM Corp. 1998, 2000

#pragma chars

Description

The *#pragma chars* directive sets the sign type of char objects to be either *signed* or *unsigned*.

C In C, this pragma must appear before any source statements, in order for this pragma to take effect.

C++ In C++ incremental mode, this pragma may be defined anywhere in the source file. In C++ batch mode, the pragma must appear before any souce statements.

Syntax

Once specified, it applies to the entire file and cannot be turned off. If a source file contains any functions that you want to be compiled without #pragma chars, place these functions in a different file. If the pragma is specified more than once in the source file, the first one will take precedence.

Note: the default character type behaves like an unsigned char.

RELATED CONCEPTS

Preprocessor Directives - An Overview

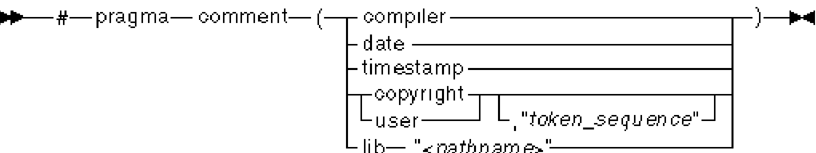
"Chapter 1. Preprocessor Directives - #pragma" on page 1

#pragma comment

Description

The *#pragma comment* directive places a comment into the target or object file.

Syntax



where:

compiler	the name and version of the compiler is appended to the end of the generated object module.
date	the date and time of compilation is appended to the end of the generated object module.
timestamp	the date and time of the last modification of the source is appended to the end of the generated object module.

copyright	the text specified by the *token_sequence* is placed by the compiler into the generated object module and is loaded into memory when the program is run.
user	the text specified by the *token_sequence* is placed by the compiler into the generated object but is *not* loaded into memory when the program is run.
OS/2 WIN lib	A library search record is placed into generated object module. Equivalent to #pragma library.

RELATED CONCEPTS

Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

#pragma define (C++)

Description

The #*pragma define* directive forces the definition of a template class without actually defining an object of the class. This pragma is only provided for backward compatibility purposes. The user can explicitly instantiate a class, function or member template specialization by using a construct of the form:

```
template declaration
```

For example:

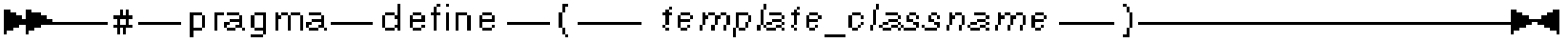
```
#pragma define(Array<char>)
```

is equivalent to:

```
template class Array<char>;
```

This pragma must be defined in global scope (i.e. it cannot be enclosed inside a function/class body).

Syntax



where the *template_classname* is the name of the template to be defined.

The pragma can only appear in the global scope. It is used when organizing your program for the efficient or automatic generation of template functions.

RELATED CONCEPTS

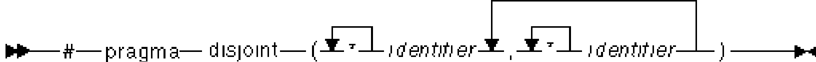
Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

#pragma disjoint

Description

The *#pragma disjoint* directive lists the identifiers that are not aliased to each other within the scope of their use.

Syntax



The directive informs the compiler that none of the identifiers listed shares the same physical storage, which provides more opportunity for optimizations. If any identifiers actually share physical storage, the pragma may cause the program to give incorrect results. This pragma directive can be enabled or disabled by specifying the opt(pragmadisjoint) option in the configuration file.

An identifier in the directive must be visible at the point in the program where the pragma appears. The identifiers in the disjoint name list cannot refer to any of the following:

- a member of a structure, or union
- a structure, union, or enumeration tag
- an enumeration constant
- a typedef name
- a label

Example

```
int a, b, *ptr_a, *ptr_b;
#pragma disjoint(*ptr_a, b) // *ptr_a never points to b
#pragma disjoint(*ptr_b, a) // *ptr_b never points to a
one_function()
{
     b = 6;
     *ptr_a = 7; // Assignment does not alter the value of b
     another_function(b); // Argument "b" has the value 6
}
```

Because external pointer *ptr_a* does not share storage with and never points to the external variable *b*, the assignment of 7 to the object that *ptr_a* points to will not change the value of *b*. Likewise, external pointer *ptr_b* does not share storage with and never points to the external variable *a*. The compiler can assume that the argument of *another_function* has the value 6 and will not reload the variable from memory.

RELATED CONCEPTS

Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1
Functionality of Configuration Files

RELATED REFERENCES

pragmadisjoint Optimization Option - opt(pragmadisjoint)

#pragma entry

C++ AIX Under the AIX platform, this pragma is not available for C, but is available for C++.

Description

The *#pragma entry* directive specifies the function to be used as the entry point for the application being built.

Syntax

The *function_name* is the name of the function to be used as an entry point. This function must be in the same compilation unit as the #pragma entry directive. The following types of functions cannot be used with #pragma entry:

- functions with C++ linkage
- C++ member functions
- template member functions
- overloaded functions
- OS/2 Under the OS/2® platform, your function must have _System linkage.

  WIN Under the Windows® platform, your function must have __cdecl linkage.

Normally when an application is started, the system calls the C library entry point. When you specify a different entry point using the #pragma entry, the system calls that entry point and does not perform any C library initialization or termination. If you use #pragma entry, you must ensure that your executable file does not require library initialization or termination, or you must provide your own initialization and termination functions.

RELATED CONCEPTS

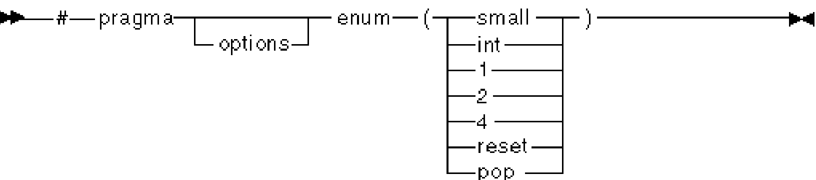
Preprocessor Directives - An Overview
“Chapter 1. Preprocessor Directives - #pragma” on page 1

#pragma enum

Description

The *#pragma enum* directive specifies the size of enum variables that follow. The size at the left brace of a declaration is the one that affects that declaration, regardless of whether further enum directives occur within the declaration. This pragma pushes a value on a stack each time it is used, with a reset option available to return to the previously pushed value.

Syntax



where *option* can be substituted with one of the following:

Options	
small	enum size is the smallest integral type that can contain all variables.
int	enum size is 4
1	enum size is 1
2	enum size is 2
4	enum size is 4
pop	the option will reset the enum size to the one before the previously set enum size.
reset	the option is an alternative method of resetting the enum size to the one before the previously set enum size. This option is provided for backwards compatibility.

Example:

```
#pragma enum(1)
#pragma enum(2)
#pragma enum(4)
#pragma enum(pop)   /* will reset enum size to 2 */
#pragma enum(reset) /* will reset enum size to 1 */
#pragma enum(pop) /* will reset enum size to default
```

Popping on an empty stack generates a warning message and the enum value remains unchanged.

The gen(enumsize) option can be used to set the enum size to default. The #pragma enum directive overrides gen(enumsize).

RELATED CONCEPTS

Preprocessor Directives - An Overview
“Chapter 1. Preprocessor Directives - #pragma” on page 1

RELATED REFERENCES

enumsize code generation option - gen(enumsize) option

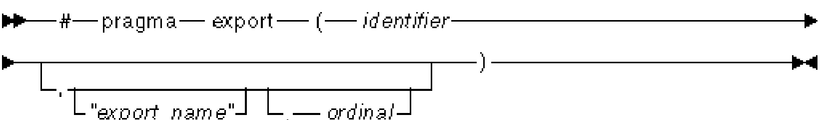
#pragma export

C++ AIX Under AIX, this pragma is not available for C, but it is available for C++

Description

The *#pragma export* directive declares that a shared library function or variable is to be exported and specifies the name of the function outside the shared library.

Syntax



where:

identifier
the name of the function or variable in the shared library.

C++ If the identifier is the name of an overloaded function or a member function, there is a risk that the pragma will override the compiler-generated names. This will create problems during linking.

export_name
The export name is the name for identifier outside of the shared library. The export name must be fully decorated according to the linkage convention that applies to it. This parameter is optional, however if we do not wish to specify the export_name but we want to specify an ordinal number, then all commas must be present. For example:

```
#pragma export(identifier,, ordinal)
```

ordinal
the number of the identifier within the DLL.

* If no *export_name* is specified, identifier is used.

RELATED CONCEPTS

Preprocessor Directives - An Overview
“Chapter 1. Preprocessor Directives - #pragma” on page 1

RELATED REFERENCES

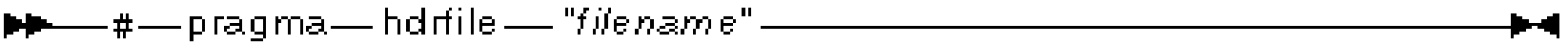
_declspec Keyword
_Export Keyword
Keywords in C and C++

#pragma hdrfile (C)

Description

The *#pragma hdrfile* directive specifies the filename of the precompiled header to be generated and/or used.

Syntax



One of the compiler options, file(genpch, ...) or file(usepch, ...), must be specified to allow more than one precompiled header to be use for a single application.

If a file name is specified both on the command line and on #pragma hdrfile, the name specified on the pragma takes precedence. If the name specified is a directory, then the the compiler searches for or generates a file with the default name in that directory.

In order to maximize the reuse of precompiled headers, the use #pragma hdrfile in combination with #pragma hdrstop to manually limit the initial sequence of #include directives.

Use precompiled header files to decrease compile time. Using precompiled headers will not improve compile time performance in most applications without some organization of the headers included by each source file.

"Examples using #pragma hdrfile (C)"

RELATED CONCEPTS

Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

RELATED REFERENCES

OS/2 WIN genpch C File Option - file(genpch, ...)

OS/2 WIN usepch C File Option - file(usepch, ...)

Examples using #pragma hdrfile (C)

```
/***********************************************************************
*  In the following example, the headers h1.h and h2.h are precompiled  *
* using the file fred.pch (provided /Si or /Fi are specified). If      *
* /Fidave.pch is specified alone, the compiler looks for the precompiled*
* headers in fred.pch but will not generate new headers.  *
***********************************************************************/
#pragma hdrfile "fred.pch"
#include "h1.h"
#include "h2.h"
main()
{
// Your Code Goes Here
}
```

```
/***********************************************************************
*   In the following example, only the header h1.h will be precompiled  *
* using the file fred.pch (provided /Si or /Fi are specified). If      *
* /Sidave.pch is specified alone, the compiler looks for the precompiled*
* headers in fred.pch but will not generate new headers.               *
***********************************************************************/
#pragma hdrfile "fred.pch"
#include "h1.h"
#pragma hdrstop
#include "h2.h"
main()
{
// Your Code Goes Here
}
```

RELATED REFERENCES

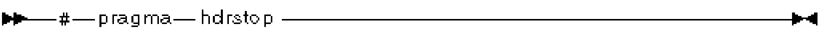
C "#pragma hdrfile (C)" on page 11

#pragma hdrstop (C)

Description

The *#pragma hdrstop* directive manually terminates the initial sequence of #include directives being considered for precompilation.

Syntax



It has no effect if:

- The initial sequence of #include directives has already ended
- Neither the file(genpch, ...) option nor the file(usepch, ...) option is specified
- It does not appear in the primary source file

Use precompiled header files to decrease compile time. Using precompiled headers will not improve compile time performance in most applications without some organization of the headers included by each source file.

"Examples using #pragma hdrstop (C)"

RELATED CONCEPTS

Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

RELATED REFERENCES

OS/2 WIN genpch C File Option - file(genpch, ...)
OS/2 WIN usepch C File Option - file(usepch, ...)

Examples using #pragma hdrstop (C)

```
/***********************************************************************
*    In the following example, only the header h1.h will be precompiled *
* using the file default.pch (provided /Si or /Fi are specified). If    *
* /Sidave.pch /Fijohn.pch are specified, the compiler will look for the *
* precompiled headers in john.pch and will regenerate them if they are  *
* not found or not usable.                                              *
***********************************************************************/
#include "h1.h"
#pragma hdrstop
#include "h2.h"
main()
{
// Your Code Goes Here
}
```

```
/***********************************************************************
* In the following example, no precompiled headers will be generated or *
* used for the compilation, even if /Fi or /Si are specified.           *
***********************************************************************/
#pragma hdrstop
#include "h1.h"
#include "h2.h"
main()
{
// Your Code Goes Here
}
```

RELATED REFERENCES

"#pragma hdrstop (C)" on page 13

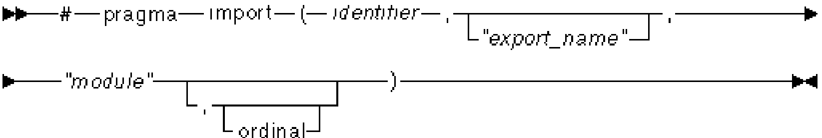
#pragma import

C++ AIX Under AIX, this pragma is not available for C, but it is available for C++.

Description

The *#pragma import* directive lets you import a function or a variable from a shared library using either an ordinal number or a name different from the one that it has in the shared library.

Syntax



AIX Ordinals are not supported in AIX; everything is by name.

The *identifier* is the name you use in your source to refer to the function or variable. The *"export_name"* is the name of the function or variable in the shared library.

C++ For C++ files, *"export_name"* can also be a function prototype. If *"export_name"* is not specified, it is assumed to be the same as *identifier*.

C++ If the identifier is the name of an overloaded function or a member function, there is a risk that the pragma will override the compiler-generated names. This will create problems during linking.

Note: Both *identifier* and *"export_name"*must be defined only once in each compilation unit.

The *"module"* is the name of the shared library containing the *identifier*, and *ordinal* indicates the position of the function or variable within the shared library. Ordinal numbers are described in more detail in the Toolkit documentation.

The information provided by #pragma import is used at load time to locate the imported *identifier*. If *ordinal* is 0, the *"export_name"* is used to find the *identifier*. If *ordinal* is any other number, *"export_name"* is ignored and the *identifier* is located by number. It is usually faster to locate the *identifier* by number than by name. By default, *ordinal* has a value of 0

Note: You cannot use the ordinals provided in the Toolkit header files with #pragma import. These ordinals are provided as C macros that cannot be used in #pragma directives.

If an identifier is specified in more than one #pragma import, all instances of the pragma must have the same *"export_name"*, *module*, and *ordinal* value. If the export name, module name, or ordinal value are different, an error will occur.

For example, the following imports "Joe" from the "TeamList":

```
#pragma import(player, "Joe", "TeamList", 2)
```

RELATED CONCEPTS

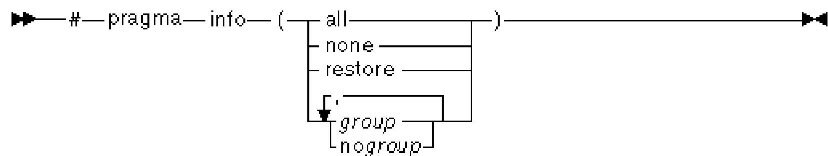
Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

#pragma info

Description

The *#pragma info* directive controls the diagnostic messages generated by the info(...) compiler option. The #pragma directive overrides any info(...) option stated in the configuration file.

Syntax:



The effects of the options used with #pragma info are listed as follows:

all	turns on all diagnostic checking.
none	turns off all diagnostic suboptions for specific portions of your program
restore	restores the options that were in effect before the previous #pragma info directive.
group	generates all messages associated with that diagnostic group
nogroup	suppresses all messages associated with that group.

For example:

```
#pragma info(ret)
```

RELATED CONCEPTS

Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

RELATED REFERENCES

Information Message Groups

C info Miscellaneous Option - info(...)

Example: #pragma isolated_call

The following example shows the use of the #pragma isolated_call directive. Because the function `this_function` does not have side effects, a call to it will not

change the value of the external variable a. The compiler can assume that the argument to other_function has the value 6 and will not reload the variable from memory.

```
int a;
// Assumed to have no side effects
int this_function(int);

#pragma isolated_call=this_function
that_function()
{
   a = 6;
   // Call does not change the value of "a"
   this_function(7);
   // Argument "a" has the value 6
   other_function(a);
}
```

The isolated call compiler option has the same effect as this pragma. The ignprag compiler option causes aliasing pragmas to be ignored. Use the ignprag compiler option to debug applications containing the #pragma isolated_call directive.

RELATED CONCEPTS

Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

RELATED REFERENCES

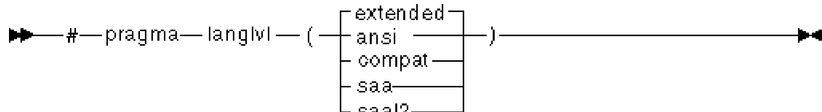
#pragma isolated_call

#pragma langlvl (C)

Description

The *#pragma langlvl* directive selects the C language level for compilation.

Syntax



This pragma can be specified only once in a source file, and it must appear before any statements in a source file. The compiler uses predefined macros in the header files to make declarations and definitions available that define the specified language level.

ansi	Defines the predefined macros __ANSI__ and __STDC__ and undefines other langlvl variables. Allows only language constructs that conform to ANSI/ISO C standards.
extended	Defines the predefined macro __EXTENDED__ and undefines other langlvl variables. The default language level is extended.
AIX classic	Defines the predefined macro __CLASSIC__ and undefines other langlvl variables.

saa	Defines the predefined macro __SAA__ and undefines other langlvl variables. Allows only language constructs that conform to the most recent level of SAA C standards (currently Level 2). These include ANSI C constructs. This language level is valid for C programs only.
saal2	Defines the predefined macro __SAAL2__ and undefines other langlvl variables. Allows only language constructs that conform to SAA Level 2 C standards. These include ANSI C constructs. This language level is valid for C programs only.

The langlvl configuration file options lang(level, ansi), lang(level, extended), lang(level, classic), lang(level, saa), and lang(level, saal2) have the same effect as this pragma.

RELATED CONCEPTS

Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

RELATED REFERENCES

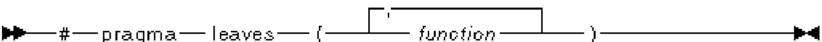
level Language Option - lang(level, ...)

#pragma leaves

Description

The *#pragma leaves* directive takes a function name and specifies that the function never returns to the instruction after the call.

Syntax



This pragma tells the compiler that the flow of the program's execution ends, although the program does not terminate, when *function* is called.

The advantage of the pragma is that it allows the compiler to ignore any code that exists after *function*, in turn, the optimizer can generate more efficient code. This pragma is commonly used for custom error-handling functions, in which programs can be terminated if a certain error is encountered. Some functions which also behave similarily are exit, longjmp, and terminate.

Example

```
#pragma leaves(handle_error_and_quit)
void test_value(int value)
{
if (value == ERROR_VALUE)
{
handle_error_and_quit(value);
```

```
TryAgain(); // optimizer ignores this because
// never returns to execute it
}
}
```

RELATED CONCEPTS

Preprocessor Directives - An Overview

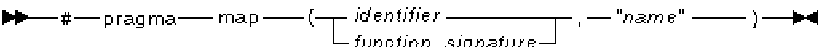
"Chapter 1. Preprocessor Directives - #pragma" on page 1

#pragma map

Description

The *#pragma map* directive tells the compiler that all references to an identifier are to be converted to *"name"*.

Syntax



The follow list describes the options available for #pragma map:

identifier
: A name of a data object or a nonoverloaded function with external linkage.

  C++ If the identifier is the name of an overloaded function or a member function, there is a risk that the pragma will override the compiler-generated names. This will create problems during linking.

function_signature
: A name of a function or operator with internal linkage. The name can be qualified.

name
: The external name that is to be bound to the given object, function, or operator.

You should not use #pragma map to map the following:
- C++ Member functions
- Overloaded functions
- Objects generated from templates
- Functions with C++ or built in linkage

The directive can appear anywhere in the program. The identifiers appearing in the directive, including any type names used in the prototype argument list, are resolved as though the directive had appeared at file scope, independent of its actual point of occurrence.

For example,

```
int func(int);
class X {
public:
void func(void);
#pragma map(func, "funcname1") // maps ::func to funcname1
#pragma map(X::func, "funcname2") // maps X::func to funcname2
};
```

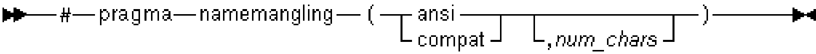
RELATED CONCEPTS
Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

#pragma namemangling (C++)

Description

The *#pragma namemangling* directive sets the maximum length for external symbol names generated from C++ source code.

Syntax



ansi	The name mangling scheme complies with the C++ standard. If you specify ansi but do not specify a size, the default maximum is 64000 characters.
compat	The name mangling scheme is the same as in earlier versions of VisualAge C++. The default maximum is 255 characters. Use this scheme for compatability with link modules created with earlier versions of VisualAge C++.

RELATED CONCEPTS
Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

#pragma object_model (C++)

Description

The *#pragma object_model* directive specifies the object model to use for the structures, unions, and classes that follow it.

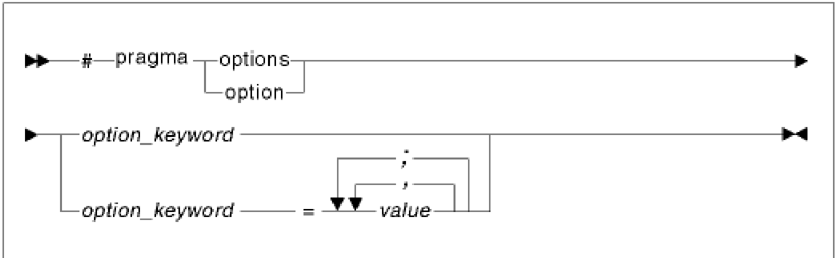
RELATED CONCEPTS
Object Models
Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1
RELATED REFERENCES
objectModel Code Generation Option (C++)

#pragma options (AIX)

Description

The *#pragma options* directive specifies compiler options for your source program.

Syntax



The following are valid *options* for #pragma options:

Valid *options*	**Description**
align=*option*	equivalent to option gen(align,*option*).
[no]ansialias	equivalent to option opt(alias,ansi).
arch=*option*	equivalent to option gen(arch,*option*).
assert=*option*	equivalent to option opt(alias,*option*).
chars=*option*	equivalent to "#pragma chars" on page 6.
[no]check	equivalent to option gen(check,zerodivide).
[no]compact	equivalent to option opt(size).
enum=*option*	equivalent to "#pragma enum" on page 9.
flag=*option*	Specify the severity level of diagnostics to be reported. #pragma options flag=i ==> "#pragma report (C++)" on page 24 (level,I) #pragma options flag=w ==> "#pragma report (C++)" on page 24 (level,W) #pragma options flag=e,s,u ==> "#pragma report (C++)" on page 24 (level,E)
[no]flttrap=*option*	equivalent to option gen(floatTrap,*option*).
float=[no]*option*	equivalent to option gen(float,*option*).
[no]fold	equivalent to option gen(float,fold).
[no]ignerrno	equivalent to option opt(ignerrno).
[no]info=*option*	equivalent to "#pragma info" on page 15.
initauto=*value*	equivalent to option gen(initauto,*value*).
[no]inlglue	equivalent to option gen(inlinePointerGlue).
isolated_call=*names*	equivalent to #pragma isolated_call.
[no]libansi	equivalent to option gen(libansi).

[no]longlong	equivalent to option lang(longlong).
[no]maxmem=*number*	equivalent to option opt(maxmem,*number*).
C++ priority=*number*	equivalent to “#pragma priority (C++)” on page 22.
[no]ro	equivalent to option gen(readOnly).
[no]roconst	equivalent to option gen(readOnlyConst).
C++ [no]rwvftable	equivalent to option WIN gen(vftReadOnly) .
spill=*number*	equivalent to option opt(registerSpillSize,*number*).
[no]strict	equivalent to option opt(strict).
tbtable=*option*	equivalent to option link(traceBackTable,*option*).
tune=*option*	equivalent to option opt(tune,*option*).
[no]unroll unroll=*number*	equivalent to option opt(loopUnroll).
C++ [no]vftable	equivalent to option gen(vft).

C The following are valid *options* for #pragma options only in C:

C **Valid** *options*	**Description**
[no]attr attr=full	Produces an attribute listing containing all names.
[no]cinc=prefix	Specifies files to be included with implicit extern “C”.
[no]dbcs	String literals and comments can contain DBCS characters.
[no]dbxextra	Generates symbol table information for unreferenced variables.
[no]digraph	Allows special digraph and keyword operators.
[no]dollar	equivalent to option lang(allowDollarInNames).
[no]extchk	Performs external name type-checking and function call checking.
[no]funcsect	Places intructions for each function in a separate cset.
[no]fullpath	Specifies the path information stored for files for dbx stabstrings.
genpcomp	Generates pre-compiled headers.
halt	Stops compiler when errors of the specified severity detected.
[no]idirfirst	Specifies search order for user include files.
langlvl	Specifies different language levels.
[no]ldbl128	gen(longdouble) is a CodeStore wide option.
list	Produces an object listing.
[no]longdouble	gen(longdouble) is a CodeStore wide option.
[no]mbcs	String literals and comments can contain DBCS characters.
macpstr	Allows ’\p’ as the first character of a string constant.
once	Processes include file only once.

showinc	Includes the source of all included files in the listing.
som (C++)	Turns on implicit SOM mode.
somgs (C++)	Disables direct access to attributes for SOM objects.
source	Produces a source listing.
srcmsg	Reconstructs source lines in error along with the diagnostic messages.
nostdinc	The include directory is not searched.
[no]upconv	equivalent to option lang(preserveUnsignedPromotion).
xref	Produces a cross-reference listing.

By default, the options specified apply to the entire source program. If you specify more than one compiler option, use a blank space to separate them.

C If certain options are to apply to the entire program, then the #pragma options directive should be specified before any statements.

Eight compiler options that can be specified with the #pragma options directive have corresponding preprocessor #pragma directives. They are:

#pragma option Name	#pragma Name
chars	"#pragma chars" on page 6
enum	"#pragma enum" on page 9
flag	C++ "#pragma report (C++)" on page 24
info	"#pragma info" on page 15
isolated_call	isolated_call
langlvl	C "#pragma langlvl (C)" on page 16
priority	C++ "#pragma priority (C++)"
ro	"#pragma strings" on page 26

RELATED CONCEPTS

Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

RELATED REFERENCES

Build Options

AIX Optimization Options

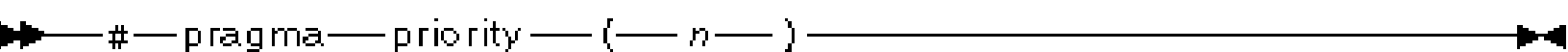
AIX Code Generation Options

#pragma priority (C++)

Description

The *#pragma priority* directive specifies the order in which static objects are to be initialized at run time.

Syntax



Where *n* is an integer literal in the range of INT_MIN to INT_MAX. The default value is 0. A negative value indicates a higher priority; a positive value indicates a lower priority. The first 1024 priorities (INT_MIN to INT_MIN + 1023) are reserved for use by the compiler and its libraries. The priority value specified applies to all runtime static initialization in the current compilation unit.

Any global object declared before another object in a file is constructed first. Use #pragma priority to specify the construction order of objects across files. However, if the user is creating an executable or shared library target from source files, VisualAge C++ will check dependency ordering, which may override #pragma priority.

For example, if the constructor to object B is passed object A as a parameter, then VisualAge C++ will arrange for A to be constructed first, even if this violates the top-to-bottom or #pragma priority ordering. This is essential for orderless programming, which VisualAge C++ permits. If the target is an .obj/.lib, this processing is not done, because there may not be enough information to detect the dependencies.

To ensure that the objects are always constructed from top to bottom in a file, the compiler enforces the restriction that the priority specifies all objects before and all objects after it until the next #pragma (is encountered) is at that priority.

For example,

```
#pragma priority(1)
```

RELATED CONCEPTS

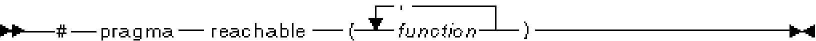
Preprocessor Directives - An Overview
“Chapter 1. Preprocessor Directives - #pragma” on page 1

#pragma reachable

Description

The *#pragma reachable* directive declares that the point after the call to a routine, *function*, can be the target of a branch from some unknown location. This pragma should be used in conjunction with setjmp.

Syntax



RELATED CONCEPTS

Preprocessor Directives - An Overview

"Chapter 1. Preprocessor Directives - #pragma" on page 1

#pragma report (C++)

Description

The *#pragma report* directive controls the generation of specific messages. The pragma will take precedence over #pragma info. #pragma report(pop) will revert the report level to the previous level. If no previous report level was specified, then a warning will be issued and the report level will remain unchanged.

Syntax

There are three ways in which this directive can be used:

```
#pragma report ( level, (E|W|I) )
#pragma report ( (enable | disable), message_number )
#pragma report (pop)
```

where:

level	indicates the minimum severity level of diagnostic messages to display.

E | W | I	used in conjunction with level to determine the type of diagnostic messages to display. **E**: signifies a minimum message severity of 'error'. This is considered as the most severe type of diagnostic message. A report level of 'E' will display only 'error' messages. An alternative way of setting the report level to 'E' is by specifying the report(level, E) compiler option. **W**: signifies a minimum message severity of 'warning'. A report level of 'W' will filter out all informational messages, and display only those messages classified as warning or error messages. An alternative way of setting the report level to 'W' is by specifying the report(level, W) compiler option. **I**: signifies a minimum message severity of 'information'. Information messages are considered as the least severe type of diagnostic message. A level of 'I' would display messages of all types. The VisualAge C++ development environment sets this as the default option. An alternative way of setting the report level to 'I' is by specifying the report(level, I) compiler option.
enable | disable	enables or disables the specified message number.
message_number	is an identifier containing the message number prefix, followed by the message number. An example of a message number is: CPPC1004
pop	resets the report level back to the previous report level. If a pop operation is performed on an empty stack, the report level will remain unchanged and no message will be generated.

Examples

```
1  #pragma info(all)
2  #pragma report(level, W)
```

In the example above, #pragma info declares all messages to be information messages. The pragma report instructs the compiler to to display only those messages with a severity of 'W' or warning messages. In this case, none of the messages will be displayed.

```
1 #pragma report(enable, CPPC1000) // enables message number CPPC1000 2
#pragma report(level, E) // display only error messages
```

In the example above, if CPPC1000 was an error message, it would be displayed. If it was any other type of diagnostic message, it would not be displayed. Changing the order of the code like so:

```
1  #pragma report(level, E)
2  #pragma report(enable, CPPC1000)
```

would yield the same result. The order in which the two lines of code appear in, does not affect the outcome. However, if the message was 'disabled', then regardless of what report level is set and order the lines of code appear in, the diagnostic message will not be displayed.

```
1  #pragma report(level, I)
2  #pragma report(enable, CPPC1000)
3  #pragma report(level, E)
4  #pragma report(pop)
```

In line 1 of the example above, the initial report level is set to 'I', causing message CPPC1000 to display regardless of the type of diagnostic message it classified as. In line 3, a new report level of 'E' is set, indicating at only messages with a severity level of 'E' will be displayed. Immediately following line 3, the current level 'E' is 'popped' and reset back to 'I'.

RELATED CONCEPTS

Preprocessor Directives - An Overview
Preprocessor Directive - #pragma

RELATED REFERENCES

#pragma info
report level Miscellaneous Option - report(level, ...)

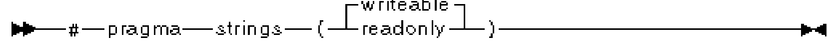
#pragma strings

Description

The *#pragma strings* directive sets storage type for strings. It specifies that the compiler can place strings into read-only memory or must place strings into read/write memory. Unlike previous versions of VisualAge C++, this pragma can appear anywhere in a source file.

C For code that will be compiled with the C compiler, this pragma must appear before any C code.

Syntax



Strings are readonly by default.

For example,

```
#pragma strings(writeable)
```

RELATED CONCEPTS

Preprocessor Directives - An Overview
"Chapter 1. Preprocessor Directives - #pragma" on page 1

Contacting IBM®

We're delighted to offer a solid cornerstone for your application development: IBM's comprehensive support services. Whether you are an occasional user with a simple question, a power user with many complex technical questions, or someone who requires application design assistance and consulting, IBM Support can meet your needs.

Comments on This Help

Please let us know about any errors or omissions in this online help or in the hardcopy Getting Started book, or our PDF documents. Send your e-mail to: torrcf@ca.ibm.com

Fee Support

Developers on the VisualAge C++ for AIX Service and Support team handle everything from simple how-to's to complex technical problems. Solutions may take the form of a brief explanation, a workaround, a fix to the current product, or a fix to the next release.

http://www.ibm.com/support/ describes IBM Support Offerings on all platforms, worldwide.

http://www.ibm.com/rs6000/support/ describes support offerings on the RS/6000® platform, in your country. It also indicates whether your country provides support electronically over the Internet in addition to telephone support.

http://www.lotus.com/passport describes the IBM and Lotus® Passport Advantage™ contracting option.

The IBM Software Support Handbook, accessible from http://www.ibm.com/software/support, also lists worldwide support contacts.

Phone numbers for information on Support Line offerings are:

- **United States**: 1-888-426-4343 (IBM Global Services), option 3 as of December 1999. Should this number change, IBM general information at 1-800-IBM-4YOU (1-800-426-4968) can route you to the appropriate group.
- **Canada**: 1-800-465-9600, option 3 as of December 1999. Should this number change, you can also contact IBM general information at 1-800-IBM-4YOU (1-800-426-4968).

Please call 1-800-266-8720 in the U.S. and Canada for information on Passport Advantage offerings.

- Elsewhere, please contact your local IBM office.

If you contact support, please have the following information available:

- The product name
- The product version
- The hardware configuration and software (product names and versions you are using)
- What happened and what you were doing when the problem occurred

© Copyright IBM Corp. 1998, 2000

- Whether you tried to solve the problem and how
- The exact wording of any messages displayed

Consulting Services

VisualAge and WebSphere™ Product Affinity Services Group is a core group of technical specialists from the IBM development labs that created the IBM VisualAge and WebSphere products. With access to a network of IBM product area experts, IBM and industry business partners, and some of the best resources in the industry, we can put the optimal team in place to meet the challenge of absorbing new technology. Our goal is to enable organizational success with VisualAge and WebSphere — ensuring that our products are used effectively within your development team.

For more information, visit http://www.ibm.com/software/ad/vaws-services/ or contact the Product Affinity Services Team at:

AIM_SERVICES@us.ibm.com