z/OS

IBM

C/C++ Messages

GC09-4819-00

z/OS

IBM

C/C++ Messages

GC09-4819-00

Note!

Before using this information and the product it supports, be sure to read the information in “Notices” on page 193.

First Edition (October 2001)

This edition applies to Version 1 Release 2 Modification 0 of z/OS C/C++ (5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure that you use the correct edition for the level of the program listed above. Also, ensure that you apply all necessary PTFs for the program.

Order publications through your IBM representative or the IBM branch office serving your location. Publications are not stocked at the address below. You can also browse the books on the World Wide Web by clicking on ″The Library″ link on the z/OS home page. The web address for this page is `http://www.ibm.com/servers/eserver/zseries/zos/bkserv`

IBM welcomes your comments. You can send your comments by mail to the following address:

IBM® Canada Ltd. Laboratory
Information Development
B3/KB7/8200/MKM
8200 Warden Avenue
Markham, Ontario Canada
L67 1C7

If you send comments, 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. About This Book

This edition of *z/OS C/C++ Messages* is intended for users of the IBM z/OS C/C++ compiler with the z/OS Language Environment® product. It provides you with information on the compiler return codes, compiler messages, utility messages, IBM Open Class® messages, and I/O Stream messages.

You may notice changes in the style and structure of some of the contents in this book; for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our books.

z/OS C/C++ and Related Publications

This section summarizes the content of the z/OS C/C++ publications and shows where to find related information in other publications.

Table 1. z/OS C/C++ Publications

Book Title and Number	Key Sections/Chapters in the Book
z/OS C/C++ Programming Guide, SC09-4765	Guidance information for: • C/C++ input and output • Debugging z/OS C programs that use input/output • Using linkage specifications in C++ • Combining C and assembler • Creating and using DLLs • Using threads in z/OS UNIX® applications • Reentrancy • Using the decimal data type in C and C++ • Handling exceptions, error conditions, and signals • Optimizing code • Optimizing your C/C++ code with Interprocedural Analysis • Network communications under z/OS UNIX • Interprocess communications using z/OS UNIX • Structuring a program that uses C++ templates • Using environment variables • Using System Programming C facilities • Library functions for the System Programming C facilities • Using run-time user exits • Using the z/OS C multitasking facility • Using other IBM products with z/OS C/C++ (CICS, CSP, DWS, DB2®, GDDM®, IMS™, ISPF, QMF) • Internationalization: locales and character sets, code set conversion utilities, mapping variant characters • POSIX character set • Code point mappings • Locales supplied with z/OS C/C++ • Charmap files supplied with z/OS C/C++ • Examples of charmap and locale definition source files • Converting code from coded character set IBM-1047 • Using built-in functions • Programming considerations for z/OS UNIX C/C++
z/OS C/C++ User's Guide, SC09-4767	Guidance information for: • z/OS C/C++ examples • Compiler options • Binder options and control statements • Specifying z/OS Language Environment run-time options • Compiling, IPA Linking, binding, and running z/OS C/C++ programs • Utilities (Object Library, DLL Rename, CXXFILT, DSECT Conversion, Code Set and Locale, ar and make, BPXBATCH) • Diagnosing problems • Cataloged procedures and REXX EXECs supplied by IBM

Table 1. z/OS C/C++ Publications (continued)

Book Title and Number	Key Sections/Chapters in the Book
C/C++ Language Reference, SC09-4815	Reference information for: • The C and C++ languages • Lexical elements of z/OS C and z/OS C++ • Declarations, expressions, and operators • Implicit type conversions • Functions and statements • Preprocessor directives • C++ classes, class members, and friends • C++ overloading, special member functions, and inheritance • C++ templates and exception handling • z/OS C and z/OS C++ compatibility
z/OS C/C++ Messages, GC09-4819	Provides error messages and return codes for the compiler, utilities, and IBM Open Class Library. For the C/C++ run-time library messages, refer to *z/OS Language Environment Run-Time Messages*, SA22-7566.
z/OS C/C++ Run-Time Library Reference, SA22-7821	Reference information for: • header files • library functions
z/OS C Curses, SA22-7820	Reference information for: • Curses concepts • Key data types • General rules for characters, renditions, and window properties • General rules of operations and operating modes • Use of macros • Restrictions on block-mode terminals • Curses functional interface • Contents of headers • The terminfo database
z/OS C/C++ Compiler and Run-Time Migration Guide, GC09-4913	Guidance and reference information for: • Common migration questions • Application executable program compatibility • Source program compatibility • Input and output operations compatibility • Class library migration considerations • Changes between releases of z/OS • C/370™ to current compiler migration • Other migration considerations
IBM Open Class Library User's Guide, SC09-4811	Guidance information for: • Using the Complex Mathematics Class Library: Review of complex numbers, header files, constructing complex objects, mathematical operators for complex, friend functions for complex, handling complex mathematics errors • Using the I/O Stream Class Library: Introduction, getting started, advanced topics, and manipulators • Using the Collection Class Library: Overview, instantiating and using, element and key functions, tailoring a collection implementation, polymorphic use of collections, support for notifications, exception handling, problem solving, compatibility with previous releases, thread safety • Using the Application Support Class Library: Introduction, String classes, Exception and Trace classes, Date and Time classes, controlling threads and protecting data, the IBM Open Class notification framework, Binary Coded (Packed) Decimal classes, text and internationalization framework, testing

Table 1. z/OS C/C++ Publications (continued)

Book Title and Number	Key Sections/Chapters in the Book
IBM Open Class Library Reference, SC09-4812	Reference information for: • Complex Mathematics Class Library • I/O Stream Class Library • Collection Class Library • Application Support Class Library
Debug Tool User's Guide and Reference, SC09-2137	Guidance and reference information for: • Preparing to debug programs • Debugging programs • Using Debug Tool in different environments • Language-specific information • Debug Tool reference
Standard C++ Library Reference, available on the z/OS C/C++ library page on the World Wide Web	The documentation, which is available at `http://www.ibm.com/software/ad/c390/czos/czosdocs.html` covers using the following three main components of the Standard C++ Library to write portable C/C++ code that complies with the ISO standards: • ISO Standard C Library • ISO Standard C++ Library • Standard Template Library (C++) The ISO Standard C++ library consists of 51 required headers. These 51 C++ library headers (along with the additional 18 Standard C headers) constitute a hosted implementation of the C++ library. Of these 51 headers, 13 constitute the Standard Template Library, or STL.
APAR and BOOKS files (Shipped with Program materials)	Partitioned data set CBC.SCCNDOC on the product tape contains the members, APAR and BOOKS, which provide additional information for using the z/OS C/C++ licensed program, including: • Isolating reportable problems • Keywords • Preparing an Authorized Program Analysis Report (APAR) • Problem identification worksheet • Maintenance on z/OS • Late changes to z/OS C/C++ publications

Note: For complete and detailed information on linking and running with z/OS Language Environment and using the z/OS Language Environment run-time options, refer to *z/OS Language Environment Programming Guide*, SA22-7561. For complete and detailed information on using interlanguage calls, refer to *z/OS Language Environment Writing Interlanguage Applications*, SA22-7563.

The following table lists the z/OS C/C++ and related publications. The table groups the publications according to the tasks they describe.

Table 2. Publications by Task

Tasks	Books
Planning, preparing, and migrating to z/OS C/C++	• *z/OS C/C++ Compiler and Run-Time Migration Guide*, GC09-4913 • *z/OS Language Environment Customization*, SA22-7564 • *z/OS Language Environment Run-Time Migration Guide*, GA22-7565 • *z/OS UNIX System Services Planning*, GA22-7800 • *z/OS Planning for Installation*, GA22-7504
Installing	• z/OS Program Directory • *z/OS Planning for Installation*, GA22-7504 • *z/OS Language Environment Customization*, SA22-7564

Table 2. Publications by Task (continued)

Tasks	Books
Coding programs	• *z/OS C/C++ Run-Time Library Reference*, SA22-7821 • *C/C++ Language Reference*, SC09-4815 • *z/OS C/C++ Programming Guide*, SC09-4765 • *z/OS Language Environment Concepts Guide*, SA22-7567 • *z/OS Language Environment Programming Guide*, SA22-7561 • *z/OS Language Environment Programming Reference*, SA22-7562 • *IBM Open Class Library User's Guide*, SC09-4811 • *IBM Open Class Library Reference*, SC09-4812
Coding and binding programs with interlanguage calls	• *z/OS C/C++ Programming Guide*, SC09-4765 • *C/C++ Language Reference*, SC09-4815 • *z/OS Language Environment Programming Guide*, SA22-7561 • *z/OS Language Environment Writing Interlanguage Applications*, SA22-7563 • *z/OS DFSMS Program Management*, SC27-1130
Compiling, binding, and running programs	• *z/OS C/C++ User's Guide*, SC09-4767 • *z/OS Language Environment Programming Guide*, SA22-7561 • *z/OS Language Environment Debugging Guide*, GA22-7560 • *z/OS DFSMS Program Management*, SC27-1130
Compiling and binding applications in the z/OS UNIX environment	• *z/OS C/C++ User's Guide*, SC09-4767 • *z/OS UNIX System Services User's Guide*, SA22-7801 • *z/OS UNIX System Services Command Reference*, SA22-7802 • *z/OS DFSMS Program Management*, SC27-1130
Debugging programs	• README file • *Debug Tool User's Guide and Reference*, SC09-2137 • *z/OS C/C++ User's Guide*, SC09-4767 • *z/OS C/C++ Messages*, GC09-4819 • *z/OS C/C++ Programming Guide*, SC09-4765 • *z/OS Language Environment Programming Guide*, SA22-7561 • *z/OS Language Environment Debugging Guide*, GA22-7560 • *z/OS Language Environment Run-Time Messages*, SA22-7566 • *z/OS UNIX System Services Messages and Codes*, SA22-7807 • *z/OS UNIX System Services User's Guide*, SA22-7801 • *z/OS UNIX System Services Command Reference*, SA22-7802 • *z/OS UNIX System Services Programming Tools*, SA22-7805 • z/OS Messages Database, available on the z/OS Library page on the World Wide Web (`http://www.ibm.com/servers/eserver/zseries/zos/bkserv`)
Using shells and utilities in the z/OS UNIX environment	• *z/OS C/C++ User's Guide*, SC09-4767 • *z/OS UNIX System Services Command Reference*, SA22-7802 • *z/OS UNIX System Services Messages and Codes*, SA22-7807
Using sockets library functions in the z/OS UNIX environment	• *z/OS C/C++ Run-Time Library Reference*, SA22-7821
Using the ISO Standard C++ Library to write portable C/C++ code that complies with ISO standards	• Standard C++ Library Reference, available on the z/OS C/C++ library page on the World Wide Web (`http://www.ibm.com/software/ad/c390/czos/czosdocs.html`)

Table 2. Publications by Task (continued)

Tasks	Books
Porting a UNIX Application to z/OS	• *z/OS UNIX System Services Porting Guide* This guide contains useful information about supported header files and C functions, sockets in z/OS UNIX, process management, compiler optimization tips, and suggestions for improving the application's performance after it has been ported. The *Porting Guide* is available as a PDF file which you can download, or as web pages which you can browse, at the following web address: `http://www-1.ibm.com/servers/eserver/zseries/zos/unix/bpxa1por.html`
Working in the z/OS UNIX System Services Parallel Environment	• *z/OS UNIX System Services Parallel Environment: Operation and Use*, SA22-7810 • *z/OS UNIX System Services Parallel Environment: MPI Programming and Subroutine Reference*, SA22-7812
Performing diagnosis and submitting an Authorized Program Analysis Report (APAR)	• *z/OS C/C++ User's Guide*, SC09-4767 • CBC.SCCNDOC(APAR) on z/OS C/C++ product tape
Tuning Large C/C++ Applications on z/OS UNIX System Services	• IBM Redbook called *Tuning Large C/C++ Applications on z/OS UNIX System Services*, which is available at: `http://www.redbooks.ibm.com/abstracts/sg245606.html`
C/C++ Applications on OS/390 UNIX	• IBM Redbook called *C/C++ Applications on OS/390 UNIX*, which is available at: `http://www.redbooks.ibm.com/abstracts/sg245992.html`
Performance considerations for XPLINK	• IBM Redbook called *XPLink: OS/390® Extra Performance Linkage*, which is available at: `http://www.redbooks.ibm.com/abstracts/sg245991.html`

Note: For information on using the prelinker, see the appendix on prelinking and linking z/OS C/C++ programs in *z/OS C/C++ User's Guide*. As of OS/390 Version 2 Release 4, this appendix contains information that was previously in the chapter on prelinking and linking z/OS C/C++ programs in *z/OS C/C++ User's Guide*. It also contains prelinker information that was previously in *z/OS C/C++ Programming Guide*.

Hardcopy Books

The following z/OS C/C++ books are available in hardcopy:

- *z/OS C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS C/C++ User's Guide*, SC09-4767
- *z/OS C/C++ Messages*, GC09-4819
- *z/OS C/C++ Programming Guide*, SC09-4765
- *z/OS C Curses*, SA22-7820
- *z/OS C/C++ Compiler and Run-Time Migration Guide*, GC09-4913
- *Debug Tool User's Guide and Reference*, SC09-2137

You can purchase these books on their own, or as part of a set. You receive *z/OS C/C++ Compiler and Run-Time Migration Guide*, GC09-4913 at no charge. Feature code 8009 includes the remaining books.

Softcopy Books

The z/OS C/C++ publications are supplied in PDF and BookMaster® formats on the following CD: *z/OS Collection*, SK3T-4269. They are also available at the following Web site:

`http://www.ibm.com/software/ad/c390/czos/czosdocs.html`

To read a PDF file, use the Adobe Acrobat Reader. If you do not have the Adobe Acrobat Reader, you can download it for free from the Adobe Web site:

`http://www.adobe.com`

To read a file in BookManager® format, use BookManager READ/MVS Version 1 Release 3 (5695-046) or the Library Reader™ for DOS, OS/2® or Windows® supplied on the CD-ROMs containing BookManager books.

If your system has BookManager Read installed, you can enter the command BOOKMGR to start BookManager and display a list of books available to you. If you know the name of the book that you want to view, you can use the OPEN command to open the book directly.

Note: If your workstation does not have graphics capability, BookManager Read cannot correctly display some characters, such as arrows and brackets.

You can also browse the books on the World Wide Web by clicking on "The Library" link on the z/OS home page. The web address for this page is:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv`

Softcopy Examples

Most of the larger examples in the following books are available in machine-readable form:
- *C/C++ Language Reference*, SC09-4815
- *z/OS C/C++ User's Guide*, SC09-4767
- *z/OS C/C++ Programming Guide*, SC09-4765
- *IBM Open Class Library User's Guide*, SC09-4811

In the following books, a label on an example indicates that the example is distributed in softcopy. The label is the name of a member in the data sets `CBC.SCCNSAM` or the directory `/usr/lpp/ioclib/sample`. The labels have the form CCN*xyyy* or CLB*xyyy*, where *x* refers to a publication:
- R and X refer to *C/C++ Language Reference*, SC09-4815
- G refers to *z/OS C/C++ Programming Guide*, SC09-4765
- U refers to *z/OS C/C++ User's Guide*, SC09-4767

Examples labelled as CCN*xyyy* appear in *C/C++ Language Reference*, *z/OS C/C++ Programming Guide*, and *z/OS C/C++ User's Guide*. Examples labelled as CLB*xyyy* appear in the *z/OS C/C++ User's Guide*. Additional IBM Open Class Samples are provided as softcopy only. They can be found in the `/usr/lpp/ioclib/sample` directory.

z/OS C/C++ on the World Wide Web

Additional information on z/OS C/C++ is available on the World Wide Web on the z/OS C/C++ home page at:

`http://www.ibm.com/software/ad/c390/czos`

This page contains late-breaking information about the z/OS C/C++ product, including the compiler, the class libraries, and utilities. It also contains a tutorial on the source level interactive debugger. There are links to other useful information, such as the z/OS C/C++ information library and the libraries of other z/OS elements that are available on the Web. The z/OS C/C++ home page also contains samples that you can download, and links to other related Web sites.

Where to find more information

Please see *z/OS Information Roadmap* for an overview of the documentation associated with z/OS, including the documentation available for z/OS Language Environment.

Accessing licensed books on the Web

z/OS licensed documentation in PDF format is available on the Internet at the IBM Resource Link Web site at:

```
http://www.ibm.com/servers/resourcelink
```

Licensed books are available only to customers with a z/OS license. Access to these books requires an IBM Resource Link Web userid and password, and a key code. With your z/OS order you received a memo that includes this key code.

To obtain your IBM Resource Link Web userid and password log on to:

```
http://www.ibm.com/servers/resourcelink
```

To register for access to the z/OS licensed books:

1. Log on to Resource Link using your Resource Link userid and password.
2. Click on **User Profiles** located on the left-hand navigation bar.
3. Click on **Access Profile.**
4. Click on **Request Access to Licensed books.**
5. Supply your key code where requested and click on the **Submit** button.

If you supplied the correct key code you will receive confirmation that your request is being processed. After your request is processed you will receive an e-mail confirmation.

Note: You cannot access the z/OS licensed books unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

To access the licensed books:

1. Log on to Resource Link using your Resource Link userid and password.
2. Click on **Library**.
3. Click on **zSeries**.
4. Click on **Software**.
5. Click on **z/OS**.
6. Access the licensed book by selecting the appropriate element.

Using LookAt to look up message explanations

LookAt is an online facility that allows you to look up explanations for z/OS messages and system abends.

Using LookAt to find information is faster than a conventional search because LookAt goes directly to the explanation.

LookAt can be accessed from the Internet or from a TSO command line.

You can use LookAt on the Internet at:

```
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookat.html
```

To use LookAt as a TSO command, LookAt must be installed on your host system. You can obtain the LookAt code for TSO from the LookAt Web site by clicking on **News and Help** or from the *z/OS Collection*, SK3T-4269.

To find a message explanation from a TSO command line, simply enter: **lookat** *message-id* as in the following example:

```
lookat iec192i
```

This results in direct access to the message explanation for message IEC192I.

To find a message explanation from the LookAt Web site, simply enter the message ID. You can select the release if needed.

Note: Some messages have information in more than one book. For example, IEC192I has routing and descriptor codes listed in *z/OS MVS Routing and Descriptor Codes*. For such messages, LookAt prompts you to choose which book to open.

Chapter 2. About IBM z/OS C/C++

The C/C++ feature of the IBM z/OS licensed program provides support for C and C++ application development on the z/OS platform. The C/C++ feature is based on the C/C++ for MVS/ESA™ product.

z/OS C/C++ includes:
- A C compiler (referred to as the z/OS C compiler)
- A C++ compiler (referred to as the z/OS C++ compiler)
- Support for a set of C++ class libraries that are available with the base z/OS operating system
- Application Support Class and Collection Class Library source
- A mainframe interactive Debug Tool (optional)
- Performance Analyzer host component, which supports the IBM C/C++ Productivity Tools for OS/390 product
- A set of utilities for C/C++ application development

IBM offers the C language on other platforms, such as the AIX®, OS/400®, VM/ESA®, and VSE/ESA™ operating systems. The AIX and OS/400 operating systems also offer the C++ language.

Changes for z/OS V1R2

z/OS C/C++ has made the following performance and usability enhancements for this release:

C++ Compiler Compliant with ISO C++ 1998 Standard

z/OS V1R2 C/C++ includes a C++ compiler which is fully compliant with the ISO C++ 1998 Standard. This includes support for:

- namespaces and associated keywords namespace and using
- new type bool and associated keywords bool, true, and false
- new class member modifying keywords mutable and explicit
- new casts and associated keywords static_cast, dynamic_cast, reinterpret_cast, and const_cast
- new template model and its associated keyword typename
- Run Time Type Identification (RTTI) and its associated keyword typeid. The C++ compiler does not support exported template definitions, nor does it allow overloading functions in ways that differ only in the linkage type of function pointer parameters.
- The C++ Standard Library, including the Standard Template Library (STL) and other library features of ISO C++ 1998

The associated run-time library DLLs use the XPLINK convention and require an XPLINK-capable run-time environment. Environments such as CICS will require the continued use of the previous environment and be compiled with the `NOXPLINK` and `TARGET(OSV2R10)` options.

Note: The OS/390 V2R10 C/C++ compiler is being shipped along with the new z/OS V1R2 C/C++ compiler. Existing C++ programs may need source code changes in order to conform to the ISO C++ 1998 Standard. If you do not require the 1998 Standard, you can use the OS/390 V2R10 compiler to avoid these source changes, but you will not get the benefits of the new features introduced in the new compiler.

IBM Open Class Library
: The IBM Open Class (IOC) is a library of C++ classes. z/OS V1R2 includes a new level of IOC, which is consistent with that shipped in VisualAge® C++ for AIX Version 5.0. This is intended to ease porting from AIX, but is not intended for use in new development. Support will be withdrawn in a future release. New application development involving C++ classes should make use of the C++ Standard Library instead of the IBM Open Class Library.

Large File Support in Standard I/O Stream Class Library
: Large file support enables access to hierarchical file system (HFS) files that are over 2 GB in size, using the C++ Standard Library.

IPA Support for XPLINK
: This feature combines the highest optimization level (IPA) for z/OS C/C++ with its high performance linkage (XPLINK).

- XPLINK is a function call linkage introduced in OS/390 V2R10 which offers significant performance increments when used in an environment of frequent calls between small functions. XPLINK makes subroutine calls more efficient by removing non-essential instructions from the main path. When all functions are compiled with the XPLINK option, function pointers can be used without restriction.
- InterProcedural Analysis (IPA) was introduced in OS/390 V1R2 C. IPA performs optimizations across compilation units, which exposes more optimization opportunities. This complements the traditional approach of optimizing within compilation units.

Enhanced ASCII Support
: z/OS V1R2 C/C++ provides enhanced ASCII support that simplifies porting of applications from ASCII platforms. It provides the ability to:

- Build ASCII-based applications by producing object code with ASCII string literals and character constants, and a flag that identifies applications as ASCII or EBCDIC
- Use Unicode-based wide characters (wchar_t) in ASCII-based applications
- Transparently call native ASCII run-time library functions from ASCII-based applications
- Process user-defined ASCII multi-byte code pages with user-supplied code set related methods
- Create ASCII-based locale objects, which allow processing of ASCII data natively at run time

The ability to produce code that contains ASCII string literals and character constants allows ASCII-dependent logic to continue working as on ASCII platforms thus eliminating the need to find all such places in the code and convert them to EBCDIC.

New Compiler Options
: z/OS V1R2 C/C++ introduces the following new compiler options:

- ASCII
- BITFIELD

- CHARS
- ENUM
- HALTONMSG (C++ only)
- KEYWORD (C++ only)
- LANGLVL (added new suboptions)
- OBJECTMODEL (C++ only)
- RTTI (C++ only)
- STATICINLINE (C++ only)
- SUPPRESS (C++ only)
- TEMPLATERECOMPILE (C++ only)
- TEMPLATEREGISTRY (C++ only)
- TMPLPARSE (C++ only)

Compiler Option Whose Syntax Has Been Changed for C++ to Match C

- INLINE: The default inline behavior has changed. In previous versions of C++, the threshold and limit values were 100 and 2000, respectively. These are now 100 and 1000.

Compiler Options That Are No Longer Supported

In z/OS V1R2 C/C++ the following compiler options are no longer supported:

- DECK: The replacement for DECK functionality that routes output to DD: SYSPUNCH is to use OBJECT(DD:SYSPUNCH). Alternatively, you can replace all references to DD:SYSPUNCH in your JCL with DD:SYSLIN, and use the OBJECT option.
- GENPCH
- HWOPTS: use ARCHITECTURE instead
- LANGLVL(COMPAT)
- OMVS: use OE instead
- SRCMSG
- SYSLIB: use SEARCH instead
- SYSPATH: use SEARCH instead
- TARGET(OSV1R2 | OSV1R3 | OSV2R4 | OSV2R5)
- USEPCH
- USERLIB: use LSEARCH instead
- USERPATH: use LSEARCH instead

Compiler Option Whose Default Value Has Changed

In z/OS V1R2 the default setting for the following compiler options has changed:

- Default is now DIGRAPH, both for C and C++
- Default is now INFO(LAN) for C++
- Default is now ROSTRING, both for C and C++

Built-in Functions for Floating-Point and Other Hardware Instructions

z/OS V1R2 has new built-in functions for floating-point and other hardware instructions, making these accessible to C/C++ programs. For information on using these built-in functions, see the appendix on built-in functions in *z/OS C/C++ Programming Guide*.

Limitations of Enhanced ASCII

This section explains under what conditions you can use Enhanced ASCII.

- A subset of C headers and functions is provided in ASCII. For more information, see *z/OS C/C++ Run-Time Library Reference*.
- The only way to get to the ASCII version of functions and the external variables `environ` and `tzname` is to use the appropriate IBM header files.
- ASCII applications may read, but not update, environment variables using the external variable. Updates to the environment variables using `environ` in an ASCII application cause unpredictable results and may result in an abend. Language Environment maintains two equivalent arrays of environment variables when running an ASCII application, one with EBCDIC encodings and the other with ASCII encodings. All ASCII compile units that use the `environ` external variable must include `<stdlib.h>` so that `environ` can be mapped to access the ASCII encoded environment strings. If `<stdlib.h>` is not included, `environ` will refer to the EBCDIC representation of the environment variable strings.

Enhanced ASCII provides limited conversion of ASCII to EBCDIC, and EBCDIC to ASCII. The character set or alphabet that is associated with any locale consists of the following:

- A common, XPG4-defined subset of characters such as POSIX portable characters
- A unique, locale-specific subset of characters such as NLS characters

The conversion only applies to the portable subset of characters that are associated with a locale. Only the EBCDIC IBM-1047 encoding of portable characters is supported.

You might encounter unexpected results in the following situations:

- If Enhanced ASCII applications are run in locales that contain non-Latin Alphabet Number 1 NLS characters, C-RTL functions might copy some of the locale's non-Latin 1 NLS characters into buffers that the application is writing to stdout or another HFS files. The non-Latin Alphabet Number 1 NLS characters would then cause problems during automatic conversion.
- Language Environment applications must select non-English message files. If your NATLANG run-time option is not UEN or ENU, messages directed to the Language Environment message file are converted to ASCII. These messages would cause problems during automatic conversion to EBCDIC.

z/OS Language Environment Downward Compatibility

z/OS Language Environment provides downward compatibility support. Assuming that you have met the required programming guidelines and restrictions, described in the *z/OS Language Environment Programming Guide*, this support enables you to develop applications on higher release levels of z/OS for use on platforms that are running lower release levels of z/OS or OS/390. In C and C++, downward compatibility support is provided through the C/C++ `TARGET` compiler option. See *z/OS C/C++ User's Guide* for details on this compiler option.

For example, a company may use z/OS V1R2 with Language Environment on a development system where applications are coded, link-edited, and tested, while using any supported lower release of OS/390 or z/OS Language Environment on their production systems where the finished application modules are used.

Downward compatibility support is not the roll-back of new function to prior releases of the operating system. Applications developed that exploit the downward compatibility support must not use any Language Environment function that is unavailable on the lower release of OS/390 or z/OS where the application will be used.

The downward compatibility support includes toleration PTFs for lower releases of OS/390 or z/OS to assist in diagnosing applications that do not meet the programming requirements for this support. (Specific PTF numbers can be found in the PSP buckets.)

The downward compatibility support provided by z/OS Language Environment and by the toleration PTFs does not change Language Environment's upward compatibility. That is, applications coded and link-edited with one release of OS/390 or z/OS Language Environment will continue to run on later releases of OS/390 or z/OS Language Environment without the need to recompile or re-link edit the application, independent of the downward compatibility support.

Downward compatibility is supported in earlier releases of OS/390 C/C++ (from Version 2 Release 6), but in OS/390 V2R6, the user is required to copy header files and link-edit `SYSLIB` data sets from the deployment release of OS/390. Starting with OS/390 Version 2 Release 10, the current level header files and `SYSLIB` can be used (the user no longer has to copy header files and `SYSLIB` data sets from the deployment release).

The C/C++ Compilers

The following sections describe the C and C++ languages and the z/OS C/C++ compilers.

The C Language

The C language is a general purpose, versatile, and functional programming language that allows a programmer to create applications quickly and easily. C provides high-level control statements and data types as do other structured programming languages. It also provides many of the benefits of a low-level language.

The C++ Language

The C++ language is based on the C language and includes all of the advantages of C listed above. In addition, C++ also supports object-oriented concepts, type genericity or templates, and an extensive library. For a detailed description of the differences between z/OS C++ and z/OS C, refer to the *C/C++ Language Reference*.

The C++ language introduces classes, which are user-defined data types that may contain data definitions and function definitions. You can use classes from established class libraries, develop your own classes, or derive new classes from existing classes by adding data descriptions and functions. New classes can inherit properties from one or more classes. Not only do classes describe the data types and functions available, but they can also hide (encapsulate) the implementation details from user programs. An object is an instance of a class.

The C++ language also provides templates and other features that include access control to data and functions, and better type checking and exception handling. It also supports polymorphism and the overloading of operators.

Common Features of the z/OS C and C++ Compilers

The C and C++ compilers, when used with z/OS Language Environment, offer many features to help your work:

- Optimization support:
  - Algorithms to take advantage of the S/390® architecture to get better optimization for speed and use of computer resources through the `OPTIMIZE` and `IPA` compiler options.
  - The `OPTIMIZE` compiler option, which instructs the compiler to optimize the machine instructions it generates to produce faster-running object code, which improves application performance at run time.
  - Interprocedural Analysis (IPA), to perform optimizations across compilation units, thereby optimizing application performance at run time.
- DLLs (dynamic link libraries) to share parts among applications or parts of applications, and dynamically link to exported variables and functions at run time.

  DLLs allow a function reference or a variable reference in one executable to use a definition located in another executable at run time. You can use both load-on-reference and load-on-demand DLLs. When your program refers to a function or variable which resides in a DLL, z/OS C/C++ generates code to load the DLL and access the functions and variables within it. This is called *load-on-reference*. Alternatively, your program can use z/OS C library functions to load a DLL and look up the address of functions and variables within it. This is called *load-on-demand*. Your application code explicitly controls load-on-demand DLLs at the source level.

  You can use DLLs to split applications into smaller modules and improve system memory usage. DLLs also offer more flexibility for building, packaging, and redistributing applications.
- Full program reentrancy

  With reentrancy, many users can simultaneously run a program. A reentrant program uses less storage if it is stored in the LPA (link pack area) or ELPA (extended link pack area) and simultaneously run by multiple users. It also reduces processor I/O when the program starts up, and improves program performance by reducing the transfer of data to auxiliary storage. z/OS C programmers can design programs that are naturally reentrant. For those programs that are not naturally reentrant, C programmers can use constructed reentrancy. To do this, compile programs with the `RENT` option and use the program management binder supplied with z/OS or the z/OS Language Environment prelinker and program management binder. The z/OS C++ compiler always ensures that C++ programs are reentrant.
- `INLINE` compiler option

  Additional optimization capabilities are available with the `INLINE` compiler option.
- Locale-based internationalization support derived from *IEEE POSIX 1003.2-1992* standard. Also derived from *X/Open CAE Specification, System Interface Definitions, Issue 4* and *Issue 4 Version 2*. This allows programmers to use locales to specify language/country characteristics for their applications.
- The ability to call and be called by other languages such as assembler, COBOL, PL/1, compiled Java™, and Fortran, to enable programmers to integrate z/OS C/C++ code with existing applications.
- Exploitation of z/OS and z/OS UNIX technology.

  z/OS UNIX is an IBM implementation of the open operating system environment, as defined in the XPG4 and POSIX standards.
- Support for the following standards at the system level:

  - A subset of the extended multibyte and wide character functions as defined by *Programming Language C Amendment 1*. This is *ISO/IEC 9899:1990/Amendment 1:1994(E)*
  - *ISO/IEC 9945-1:1990(E)/IEEE POSIX 1003.1-1990*
  - A subset of *IEEE POSIX 1003.1a, Draft 6, July 1991*
  - *IEEE Portable Operating System Interface (POSIX) Part 2, P1003.2*
  - A subset of *IEEE POSIX 1003.4a, Draft 6, February 1992* (the IEEE POSIX committee has renumbered POSIX.4a to POSIX.1c)
  - *X/Open CAE Specification, System Interfaces and Headers, Issue 4 Version 2*
  - A subset of *IEEE 754-1985 (R1990) IEEE Standard for Binary Floating-Point Arithmetic (ANSI)*, as applicable to the S/390 environment.
  - *X/Open CAE Specification, Network Services, Issue 4*
- Year 2000 support
- Support for the Euro currency

z/OS C Compiler Specific Features

In addition to the features common to z/OS C and C++, the z/OS C compiler provides you with the following capabilities:

- The ability to write portable code that supports the following standards:
  - All elements of the ISO standard *ISO/IEC 9899:1990 (E)*
  - *ANSI/ISO 9899:1990[1992]* (formerly *ANSI X3.159-1989 C*)
  - *X/Open Specification Programming Language Issue 3, Common Usage C*
  - *FIPS-160*
- System programming capabilities, which allow you to use z/OS C in place of assembler
- Extensions of the standard definitions of the C language to provide programmers with support for the z/OS environment, such as fixed-point (packed) decimal data support

z/OS C++ Compiler Specific Features

In addition to the features common to z/OS C and C++, the z/OS C++ compiler supports the *International Standard for the C++ Programming Language (ISO/IEC 14882:1998)* specification.

Class Libraries

z/OS V1R2 C/C++ provides the following class libraries, which are all thread-safe:

- C++ Standard Library, including the Standard Template Library (STL) and other library features of ISO C++ 1998
- IBM Open Class Library for z/OS V1R2
- IBM Open Class Library for OS/390 V2R10

Refer to *z/OS C/C++ Compiler and Run-Time Migration Guide* and *IBM Open Class Library User's Guide* for more details on the components of these libraries.

For new code and the most portable code you will want to use the new C++ Standard Library, which includes the following:

- The C++ Standard I/O Stream Library for performing input and output (I/O) operations

- The C++ Standard Complex Mathematics Library for manipulating complex numbers
- The Standard Template Library (STL) which is composed of C++ template-based algorithms, container classes, iterators, localization objects, and the string class

The IBM Open Class (IOC) is a comprehensive library of C++ classes that you can use to develop applications. z/OS V1R2 includes a new level of IOC which is consistent with that shipped in VisualAge C++ for AIX V5.0. This is intended to ease porting from AIX, but is not intended for use in new development. Support will be withdrawn in a future release.

The z/OS V1R2 IBM Open Class Library includes:

- The Application Support Class Library which provides the basic abstractions that are needed during the creation of most C++ applications, including String, Date, Time, and Decimal. The Application Support Class Library corresponds to the IOC member in the data sets.
- The Collection Class Library implements a wide variety of classical data structures such as stack, tree, list, hash table, and so on. The Collection Class Library provides developers with a consistent set of building blocks from which they can derive application objects. The library design exploits features of the C++ language such as exception handling and template support. The Collection Class Library corresponds to the COLL member in the data sets.

The z/OS V1R2 IBM Open Class enables you to choose between the C++ Standard I/O Stream and Complex Mathematics libraries, and the UNIX Systems Laboratories C++ Language System Release (USL) I/O Stream and Complex Mathematics libraries.

The OS/390 V2R10 IBM Open Class Library and USL class libraries include the following:

- The USL I/O Stream Class Library (corresponds to the IOSTREAM member in the data sets)
- The USL Complex Mathematics Class Library (corresponds to the COMPLEX member in the data sets)
- The Application Support Class Library (corresponds to the APPSUPP member in the data sets)
- The Collection Class Library (corresponds to the COLLECT member in the data sets)

Note: Retroactive to OS/390 Version 1 Release 3, the IBM Open Class Library is licensed with the base operating system. This enables applications to use this library at run time without having to license the z/OS C/C++ compiler features or to use the DLL Rename Utility.

IBM Open Class Library Source

The IBM Open Class Library Source consists of the following:

- Application Support Class Library source code
- Collection Class Library source code

Utilities

The z/OS C/C++ compilers provide the following utilities:

- The CXXFILT utility to map z/OS C++ mangled names to the original source.

- The DSECT Conversion Utility to convert descriptive assembler DSECTs into z/OS C/C++ data structures.
- The localedef utility to read the locale definition file and produce a locale object that the locale-specific library functions can use.
- The makedepend utility to derive all dependencies in the source code and write these into the makefile for the make command to determine which source files to recompile, whenever a dependency has changed. This frees the user from manually monitoring such changes in the source code.

z/OS Language Environment provides the following utilities:
- The Object Library Utility (C370LIB) to update partitioned data set (PDS and PDSE) libraries of object modules and Interprocedural Analysis (IPA) object modules.
- The DLL Rename Utility to make selected DLLs a unique component of the applications with which they are packaged. The DLL Rename Utility does not support XPLINK.
- The prelinker which combines object modules that comprise a z/OS C/C++ application, to produce a single object module. The prelinker supports only object and extended object format input files, and does not support GOFF.

The Debug Tool

z/OS C/C++ supports program development by using the Debug Tool. This optionally available tool allows you to debug applications in their native host environment, such as CICS/ESA®, IMS/ESA®, DB2, and so on. The Debug Tool provides the following support and function:
- Step mode
- Breakpoints
- Monitor
- Frequency analysis
- Dynamic patching

You can record the debug session in a log file, and replay the session. You can also use the Debug Tool to help capture test cases for future program validation, or to further isolate a problem within an application.

You can specify either data sets or hierarchical file system (HFS) files as source files.

Note: You can also use the `dbx` shell command to debug programs, as described in *z/OS UNIX System Services Command Reference*.

For further information, see “IBM C/C++ Productivity Tools for OS/390”.

IBM C/C++ Productivity Tools for OS/390

With the IBM C/C++ Productivity Tools for OS/390 product, you can expand your z/OS application development environment out to the workstation, while remaining close to your familiar host environment. IBM C/C++ Productivity Tools for OS/390 includes the following workstation-based tools to increase your productivity and code quality:
- A Performance Analyzer to help you analyze, understand, and tune your C and C++ applications for improved performance

- A Distributed Debugger that allows you to debug C or C++ programs from the convenience of the workstation
- A workstation-based editor to improve the productivity of your C and C++ source entry
- Advanced online help, with full text search and hypertext topics as well as printable, viewable, and searchable Portable Document Format (PDF) documents

In addition, IBM C/C++ Productivity Tools for OS/390 includes the following host components:

- Debug Tool
- Host Performance Analyzer

Use the Performance Analyzer on your workstation to graphically display and analyze a profile of the execution of your host z/OS C or C++ application. Use this information to time and tune your code so that you can increase the performance of your application.

Use the Distributed Debugger to debug your z/OS C or C++ application remotely from your workstation. Set a break point with the simple click of the mouse. Use the windowing capabilities of your workstation to view multiple segments of your source and your storage, while monitoring a variable at the same time.

Use the workstation-based editor to quickly develop C and C++ application code that runs on z/OS. Context-sensitive help information is available to you when you need it.

References to *Performance Analyzer* in this document refer to the IBM OS/390 Performance Analyzer included in the C/C++ Productivity Tools for OS/390 product.

z/OS Language Environment

z/OS C/C++ exploits the C/C++ run-time environment and library of run-time services available with z/OS Language Environment (formerly OS/390 Language Environment, Language Environment for MVS™ & VM, Language Environment/370 and LE/370).

z/OS Language Environment consists of four language-specific run-time libraries, and Base Routines and Common Services, as shown below. z/OS Language Environment establishes a common run-time environment and common run-time services for language products, user programs, and other products.

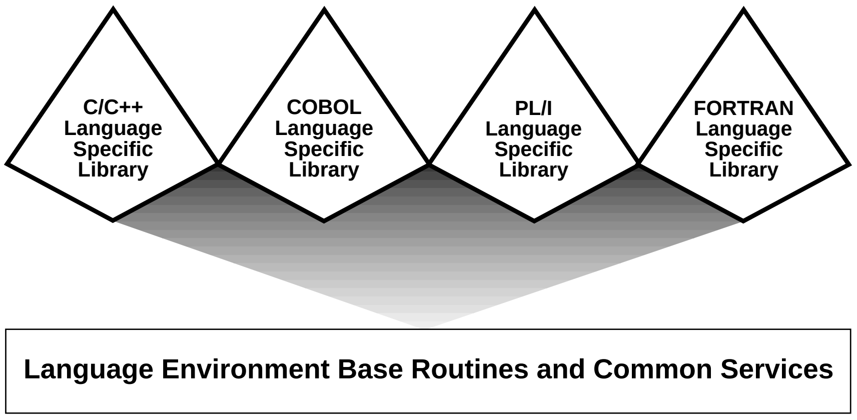
C/C++ Language Specific Library
COBOL Language Specific Library
PL/I Language Specific Library
FORTRAN Language Specific Library
Language Environment Base Routines and Common Services

Figure 1. Libraries in z/OS Language Environment

The common execution environment is composed of data items and services that are included in library routines available to an application that runs in the environment. The z/OS Language Environment provides a variety of services:

- Services that satisfy basic requirements common to most applications. These include support for the initialization and termination of applications, allocation of storage, interlanguage communication (ILC), and condition handling.
- Extended services that are often needed by applications. z/OS C/C++ contains these functions within a library of callable routines, and include interfaces to operating system functions and a variety of other commonly used functions.
- Run-time options that help in the execution, performance, and diagnosis of your application.
- Access to operating system services; z/OS UNIX services are available to an application programmer or program through the z/OS C/C++ language bindings.
- Access to language-specific library routines, such as the z/OS C/C++ library functions.

About Prelinking, Linking, and Binding

When describing the process to build an application, this document refers to the *bind step*.

Normally the Program Management Binder is used to perform the bind step. However, in many cases the prelink and link steps can be used in place of the bind step. When they cannot be substituted, and the Program Management binder alone must be used, it will be stated. For more information on the bind, prelink, and link steps, refer to *z/OS C/C++ User’s Guide*.

The terms *bind* and *link* have multiple meanings.

- With respect to building an application:

  In both instances, the program management binder is performing the actual processing of converting the object file(s) into the application executable module.

  Object files with longname symbols, reentrant writable static symbols, and DLL-style function calls require additional processing to build global data for the application.

  The term *link* refers to the case where the binder does not perform this additional processing, due to one of the following:

  - The processing is not required, because none of the object files in the application use constructed reentrancy, use long names, are DLL or are C++.
  - The processing is handled by executing the prelinker step before running the binder.

  The term *bind* refers to the case where the binder is required to perform this processing.
- With respect to executing code in an application:

  The `linkage` definition refers to the program call linkage between program functions and methods. This includes the passing of control and parameters. Refer to *C/C++ Language Reference* for more information on linkage specification.

  Some platforms have a single linkage convention. S/390 has a number of linkage conventions, including standard operating system linkage, Extra Performance Linkage (XPLINK), and different non-XPLINK linkage conventions for C and C++.

Notes on the Prelinking Process

Note that you cannot use the prelinker if you are using the `XPLINK` or `GOFF` compiler options. Also, IBM recommends using the binder without the prelinker whenever possible.

Prior to OS/390 V2R4 C/C++, the *z/OS C/C++ User's Guide* showed how to use the prelinker and linkage editor. Sections throughout the book discussed concepts of *prelinking* and *linking*. The prelinker was designed to process long names and support constructed reentrancy in earlier versions of the C complier on the MVS and OS/390 operating systems. The prelinker, shipped with the z/OS C/C++ run-time library, provides output that is compatible with the linkage editor, that is shipped with the binder.

The *binder* is designed to include the function of the prelinker, the linkage editor, the loader, and a number of APIs to manipulate the program object. Thus, the binder is a superset of the linkage editor. Its functionality provides a high level of compatibility with the prelinker and linkage editor, but provides additional functionality in some areas. Generally, the terms *binding* and *linking* are interchangeable. For more information on the compatibility between the binder, and the linker and prelinker, see *z/OS DFSMS Program Management*.

Updates to the prelinking, linkage-editing, and loading functions that are performed by the binder are delivered through the binder. If you use the prelinker shipped with the z/OS C/C++ run-time library and the linkage editor (supplied through the binder) you have to apply the latest maintenance for the run-time library as well as the binder.

If you still need to use the prelinker and linkage editor, see *z/OS C/C++ User's Guide*.

File Format Considerations

You can use the binder in place of the prelinker and linkage editor but there are exceptions, which are file format considerations. For further information, on when you cannot use the binder, see the chapter about binding z/OS C/C++ programs in the *z/OS C/C++ User's Guide*.

The Program Management Binder

The binder provided with z/OS combines the object modules, load modules, and program objects comprising an application. It produces a single z/OS output program object or load module that you can load for execution. The binder supports all C and C++ code, provided that you store the output program in a PDSE (Partitioned Data Set Extended) member or an HFS file.

If you cannot use a PDSE member or HFS file, and your program contains C++ code, or C code that is compiled with any of the `RENT`, `LONGNAME`, `DLL` or `IPA` compiler options, you must use the prelinker. C and C++ code compiled with the `GOFF` or `XPLINK` compiler options cannot be processed by the prelinker.

Using the binder without using the prelinker has the following advantages:

- Faster rebinds when recompiling and rebinding a few of your source files
- Rebinding at the single compile unit level of granularity (except when you use the `IPA` compile-time option)
- Input of object modules, load modules, and program objects
- Improved long name support:
  – Long names do not get converted into prelinker generated names
  – Long names appear in the binder maps, enabling full cross-referencing
  – Variables do not disappear after prelink
  – Fewer steps in the process of producing your executable program

Using the binder without using the prelinker has the following disadvantage:

- Long name maximum symbol length:
  – Long names currently processed by the binder are limited to 1024 characters. The prelinker supports up to (32 K - 1) characters. IBM intends to bring the binder limit in line with the prelinker in a future release.

The prelinker provided with z/OS Language Environment combines the object modules comprising a z/OS C/C++ application and produces a single object module. You can link-edit the object module into a load module (which is stored in a PDS), or bind it into a load module or a program object (which is stored in a PDS, PDSE, or HFS file).

Note: For further information on the binder, refer to the DFSMS home page at http://www.ibm.com/storage/software/sms/smshome.htm.

z/OS UNIX System Services (z/OS UNIX)

z/OS UNIX provides capabilities under z/OS to make it easier to implement or port applications in an open, distributed environment. z/OS UNIX Services are available to z/OS C/C++ application programs through the C/C++ language bindings available with z/OS Language Environment.

Together, the z/OS UNIX System Services, z/OS Language Environment, and z/OS C/C++ compilers provide an application programming interface that supports industry standards.

z/OS UNIX provides support for both existing z/OS applications and new z/OS UNIX applications through the following:

- C programming language support as defined by ISO C
- C++ programming language support as defined by ISO C++

- C language bindings as defined in the IEEE 1003.1 and 1003.2 standards; subsets of the draft 1003.1a and 1003.4a standards; *X/Open CAE Specification: System Interfaces and Headers, Issue 4, Version 2*, which provides standard interfaces for better source code portability with other conforming systems; and *X/Open CAE Specification, Network Services, Issue 4*, which defines the X/Open UNIX descriptions of sockets and X/Open Transport Interface (XTI)
- z/OS UNIX Extensions that provide z/OS-specific support beyond the defined standards
- The z/OS UNIX Shell and Utilities feature, which provides:
  - A shell, based on the Korn Shell and compatible with the Bourne Shell
  - A shell, `tcsh`, based on the C shell, `csh`
  - Tools and utilities that support the *X/Open Single UNIX Specification*, also known as *X/Open Portability Guide (XPG) Version 4, Issue 2*, and provide z/OS support. The following is a partial list of utilities that are included:

ar	Creates and maintains library archives
BPXBATCH	Allows you to submit batch jobs that run shell commands, scripts, or z/OS C/C++ executable files in HFS files from a shell session
c89	Compiles, assembles, and binds z/OS UNIX C/C++ and assembler applications
dbx	Provides an environment to debug and run programs
gencat	Merges the message text source files message file (usually *.msg) into a formatted message catalog file (usually *.cat)
iconv	Converts characters from one code set to another
lex	Automatically writes large parts of a lexical analyzer based on a description that is supplied by the programmer
localedef	Creates a compiled locale object
make	Helps you manage projects containing a set of interdependent files, such as a program with many z/OS source and object files, keeping all such files up to date with one another
yacc	Allows you to write compilers and other programs that parse input according to strict grammar rules

  - Support for other utilities such as:

c++	Compiles, assembles, and binds z/OS UNIX C++ applications
mkcatdefs	Preprocesses a message source file for input to the gencat utility
runcat	Invokes mkcatdefs and pipes the message catalog source data (the output from mkcatdefs) to gencat
dspcat	Displays all or part of a message catalog
dspmsg	Displays a selected message from a message catalog

- The z/OS UNIX Debugger feature, which provides the dbx interactive symbolic debugger for z/OS UNIX applications
- Access to a hierarchical file system (HFS), with support for the POSIX.1 and XPG4 standards
- z/OS C/C++ I/O routines, which support using HFS files, standard z/OS data sets, or a mixture of both

- Application threads (with support for a subset of POSIX.4a)
- Support for z/OS C/C++ DLLs

z/OS UNIX offers program portability across multivendor operating systems, with support for POSIX.1, POSIX.1a (draft 6), POSIX.2, POSIX.4a (draft 6), and XPG4.2.

For application developers who have worked with other UNIX environments, the z/OS UNIX Shell and Utilities are a familiar environment for C/C++ application development. If you are familiar with existing MVS development environments, you may find that the z/OS UNIX environment can enhance your productivity. Refer to *z/OS UNIX System Services User's Guide* for more information on the Shell and Utilities.

z/OS C/C++ Applications with z/OS UNIX C/C++ Functions

All z/OS UNIX C functions are available at all times. In some situations, you must specify the `POSIX(ON)` run-time option. This is required for the POSIX.4a threading functions, and the system() and signal handling functions where the behavior is different between POSIX/XPG4 and ISO. Refer to *z/OS C/C++ Run-Time Library Reference* for more information about requirements for each function.

You can invoke a z/OS C/C++ program that uses z/OS UNIX C functions using the following methods:

- Directly from a shell.
- From another program, or from a shell, using one of the `exec` family of functions, or the BPXBATCH utility from TSO or MVS batch.
- Using the POSIX `system()` call.
- Directly through TSO or MVS batch without the use of the intermediate BPXBATCH utility. In some cases, you may require the `POSIX(ON)` run-time option.

Input and Output

The C/C++ run-time library that supports the z/OS C/C++ compiler supports different input and output (I/O) interfaces, file types, and access methods. The C++ I/O Stream Class Library provides additional support.

I/O Interfaces

The C/C++ run-time library supports the following I/O interfaces:

C Stream I/O
: This is the default and the ISO-defined I/O method. This method processes all input and output on a per-character basis.

Record I/O
: The library can also process your input and output by record. A record is a set of data that is treated as a unit. It can also process VSAM data sets by record. Record I/O is a z/OS C/C++ extension to the ISO standard.

TCP/IP Sockets I/O
: z/OS UNIX provides support for an enhanced version of an industry-accepted protocol for client/server communication that is known as *sockets*. A set of C language functions provides support for z/OS UNIX sockets. z/OS UNIX sockets correspond closely to the sockets used by UNIX applications that use the Berkeley Software Distribution (BSD) 4.3

standard (also known as OE sockets). The slightly different interface of the X/Open CAE Specification, Networking Services, Issue 4, is supplied as an additional choice. This interface is known as X/Open Sockets.

The z/OS UNIX socket application program interface (API) provides support for both UNIX domain sockets and Internet domain sockets. UNIX domain sockets, or *local sockets*, allow interprocess communication within z/OS, independent of TCP/IP. Local sockets behave like traditional UNIX sockets and allow processes to communicate with one another on a single system. With Internet sockets, application programs can communicate with each other in the network using TCP/IP.

In addition, the C++ I/O Stream libraries support formatted I/O in C++. You can code sophisticated I/O statements easily and clearly, and define input and output for your own data types. This helps improve the maintainability of programs that use input and output.

File Types

In addition to conventional files, such as sequential files and partitioned data sets, the C/C++ run-time library supports the following file types:

Virtual Storage Access Method (VSAM) Data Sets

z/OS C/C++ has native support for three types of VSAM data organization:

- Key-sequenced data sets (KSDS). Use KSDS to access a record through a key within the record. A key is one or more consecutive characters that are taken from a data record that identifies the record.
- Entry-sequenced data sets (ESDS). Use ESDS to access data in the order it was created (or in reverse order).
- Relative-record data sets (RRDS). Use RRDS for data in which each item has a particular number (for example, a telephone system where a record is associated with each telephone number).

For more information on how to perform I/O operations on these VSAM file types, see *z/OS C/C++ Programming Guide*.

Hierarchical File System Files

z/OS C/C++ recognizes Hierarchical File System (HFS) file names. The name specified on the `fopen()` or `freopen()` call has to conform to certain rules (described in *z/OS C/C++ Programming Guide*). You can create regular HFS files, special character HFS files, or FIFO HFS files. You can also create links or directories.

Memory Files

Memory files are temporary files that reside in memory. For improved performance, you can direct input and output to memory files rather than to devices. Since memory files reside in main storage and only exist while the program is executing, you primarily use them as work files. You can access memory files across load modules through calls to non-POSIX `system()` and C `fetch()`; they exist for the life of the root program. Standard streams can be redirected to memory files on a non-POSIX `system()` call using command line redirection.

Hiperspace™ Expanded Storage

Large memory files can be placed in Hiperspace expanded storage to free up some of your home address space for other uses. Hiperspace expanded storage or high performance space is a range of up to 2 GB of contiguous virtual storage space. A program can use this storage as a buffer (1 gigabyte(GB) = 2^{30} bytes).

Additional I/O Features

z/OS C/C++ provides additional I/O support through the following features:

- Large file support, which enables I/O to and from hierarchical file system (HFS) files that are larger than 2 GB
- User error handling for serious I/O failures (SIGIOERR)
- Improved sequential data access performance through enablement of the DFSMS/MVS® support for 31-bit sequential data buffers and sequential data striping on extended format data sets
- Full support of PDSEs on z/OS (including support for multiple members opened for write)
- Overlapped I/O support under z/OS (NCP, BUFNO)
- Multibyte character I/O functions
- Fixed-point (packed) decimal data type support in formatted I/O functions
- Support for multiple volume data sets that span more than one volume of DASD or tape
- Support for Generation Data Group I/O

The System Programming C Facility

The System Programming C (SPC) facility allows you to build applications that require no dynamic loading of z/OS Language Environment libraries. It also allows you to tailor your application for better utilization of the the low-level services available on your operating system. SPC offers a number of advantages:

- You can develop applications that can be executed in a customized environment rather than with z/OS Language Environment services. Note that if you do not use z/OS Language Environment services, only some built-in functions and a limited set of C/C++ run-time library functions are available to you.
- You can substitute the z/OS C language in place of assembler language when writing system exit routines, by using the interfaces that are provided by SPC.
- SPC lets you develop applications featuring a user-controlled environment, in which a z/OS C environment is created once and used repeatedly for C function execution from other languages.
- You can utilize co-routines, by using a two-stack model to write application service routines. In this model, the application calls on the service routine to perform services independent of the user. The application is then suspended when control is returned to the user application.

Interaction with Other IBM Products

When you use z/OS C/C++, you can write programs that utilize the power of other IBM products and subsystems:

- Cross System Product (CSP)

  Cross System Product/Application Development (CSP/AD) is an application generator that provides ways to interactively define, test, and generate application programs to improve productivity in application development. Cross System Product/Application Execution (CSP/AE) takes the generated program and executes it in a production environment.

  Note: You cannot compile CSP applications with the z/OS C++ compiler. However, your z/OS C++ program can use interlanguage calls (ILC) to call z/OS C programs that access CSP.

- Customer Information Control System (CICS)

  You can use the CICS/ESA Command-Level Interface to write C/C++ application programs. The CICS® Command-Level Interface provides data, job, and task management facilities that are normally provided by the operating system.

  Note: Code preprocessed with CICS/ESA versions prior to V4R1 is not supported for z/OS C++ applications. z/OS C++ code preprocessed on CICS/ESA V4R1 cannot run under CICS/ESA V3R3.
- DB2 Universal Database™ (UDB) for z/OS

  DB2 programs manage data that is stored in relational databases. You can access the data by using a structured set of queries that are written in Structured Query Language (SQL).

  A DB2 program uses SQL statements that are embedded in the application program. The SQL translator (DB2 preprocessor) translates the embedded SQL into host language statements, which are then compiled by the z/OS C/C++ compilers. The DB2 program processes requests, then returns control to the application program.
- Data Window Services (DWS)

  The Data Window Services (DWS) part of the Callable Services Library allows your C or C++ program to manipulate temporary data objects that are known as TEMPSPACE and VSAM linear data sets.
- Information Management System (IMS)

  The Information Management System/Enterprise Systems Architecture (IMS/ESA) product provides support for hierarchical databases.
- Interactive System Productivity Facility (ISPF)

  z/OS C/C++ provides access to the Interactive System Productivity Facility (ISPF) Dialog Management Services. A dialog is the interaction between a user and a computer. The dialog interface contains display, variable, message, and dialog services as well as other facilities that are used to write interactive applications.
- Graphical Data Display Manager (GDDM)

  GDDM provides a comprehensive set of functions to display and print applications most effectively:
  - A windowing system that the user can tailor to display selected information
  - Support for presentation and keyboard interaction
  - Comprehensive graphics support
  - Fonts (including support for the double-byte character set)
  - Business image support
  - Saving and restoring graphic pictures
  - Support for many types of display terminals, printers, and plotters
- Query Management Facility (QMF)

  z/OS C supports the Query Management Facility (QMF), a query and report writing facility, which allows you to write applications through a callable interface. You can create applications to perform a variety of tasks, such as data entry, query building, administration aids, and report analysis.
- z/OS Java Support

  The Java language supports the Java Native Interface (JNI) for making calls to and from C/C++. These calls do not use ILC support but rather the Java defined interface JNI. Java code, which has been compiled using the High Performance

Compiler for Java (HPCJ), will support the JNI interface. Calls to C or C++ do not distinguish between compiled Java and interpreted Java.

Additional Features of z/OS C/C++

Feature	Description
long long Data Type	The z/OS C/C++ compiler supports long long as a native data type when the compiler option `LANGLVL(LONGLONG)` is turned on. This option is turned on by default by the compiler option `LANGLVL(EXTENDED)`.
Multibyte Character Support	z/OS C/C++ supports multibyte characters for those national languages such as Japanese whose characters cannot be represented by a single byte.
Wide Character Support	Multibyte characters can be normalized by z/OS C library functions and encoded in units of one length. These normalized characters are called wide characters. Conversions between multibyte and wide characters can be performed by string conversion functions such as `wcstombs()`, `mbstowcs()`, `wcsrtombs()`, and `mbsrtowcs()`, as well as the family of wide-character I/O functions. Wide-character data can be represented by the `wchar_t` data type.
Extended Precision Floating-Point Numbers	z/OS C/C++ provides three S/390 floating-point number data types: single precision (32 bits), declared as `float`; double precision (64 bits), declared as `double`; and extended precision (128 bits), declared as `long double`. Extended precision floating-point numbers give greater accuracy to mathematical calculations. As of Release 6, z/OS C/C++ also supports IEEE 754 floating-point representation. By default, `float`, `double`, and `long double` values are represented in IBM S/390 floating point format. However, the IEEE 754 floating-point representation is used if you specify the `FLOAT(IEEE754)` compile option. For details on this support, see the description of the `FLOAT` option in *z/OS C/C++ User's Guide*.
Command Line Redirection	You can redirect the standard streams `stdin`, `stderr`, and `stdout` from the command line or when calling programs using the `system()` function.
National Language Support	z/OS C/C++ provides message text in either American English or Japanese. You can dynamically switch between these two languages.
Locale Definition Support	z/OS C/C++ provides a locale definition utility that supports the creation of separate files of internationalization data, or locales. Locales can be used at run time to customize the behavior of an application to national language, culture, and coded character set (code page) requirements. Locale-sensitive library functions, such as `isdigit()`, use this information.
Coded Character Set (Code Page) Support	The z/OS C/C++ compiler can compile C/C++ source written in different EBCDIC code pages. In addition, the `iconv` utility converts data or source from one code page to another.
Selected Built-in Library Functions	Selected library functions, such as string and character functions, are built into the compiler to improve performance execution. Built-in functions are compiled into the executable, and no calls to the library are generated.
Multi-threading	Threads are efficient in applications that allow them to take advantage of any underlying parallelism available in the host environment. This underlying parallelism in the host can be exploited either by forking a process and creating a new address space, or by using multiple threads within a single process. For more information, refer to the *z/OS C/C++ Programming Guide*

Feature	Description
Multitasking Facility (MTF)	Multitasking is a mode of operation where your program performs two or more tasks at the same time. z/OS C provides a set of library functions that perform multitasking. These functions are known as the Multitasking Facility (MTF). MTF uses the multitasking capabilities of z/OS to allow a single z/OS C application program to use more than one processor of a multiprocessing system simultaneously. **Note:** `XPLINK` is not supported in an MTF environment. You can also use threads to perform multitasking with or without `XPLINK`, as described in the *z/OS C/C++ Programming Guide*.
Packed Structures and Unions	z/OS C provides support for packed structures and unions. Structures and unions may be packed to reduce the storage requirements of an z/OS C program or to define structures that are laid out according to COBOL or PL/I structure layout rules.
Fixed-point (Packed) Decimal Data	z/OS C supports fixed-point (packed) decimal as a native data type for use in business applications. The packed data type is similar to the COBOL data type `COMP-3` or the PL/I data type `FIXED DEC`, with up to 31 digits of precision. The Application Support Class Library provides the Binary Coded Decimal Class for C++ programs.
Long Name Support	For portability, external names can be mixed case and up to 1024 characters in length. For C++, the limit applies to the mangled version of the name.
System Calls	You can call commands or executable modules using the `system()` function under z/OS, z/OS UNIX, and TSO. You can also use the `system()` function to call EXECs on z/OS and TSO, or Shell scripts using z/OS UNIX.
Exploitation of ESA	Support for z/OS, IMS/ESA, Hiperspace expanded storage, and CICS/ESA allows you to exploit the features of the ESA.
Exploitation of hardware	Use the `ARCHITECTURE` compiler option to select the minimum level of machine architecture on which your program will run. `ARCH(2)` instructs the compiler to generate faster instruction sequences that are available only on newer machines. `ARCH(3)` also generates these faster instruction sequences and enables support for IEEE 754 Binary Floating-Point instructions. Code compiled with `ARCH(2)` runs on G2, G3, G4, and 2003 processors and code compiled with `ARCH(3)` runs on a G5 or G6 processor, and follow-on models. Use the `TUNE` compiler option to optimize your application for a specific machine architecture. `TUNE` impacts performance only; it does not impact the processor model on which you will be able to run your application. `TUNE(3)` optimizes your application for the newer G4, G5, and G6 processors. `TUNE(2)` optimizes your application for other architectures. For more information, refer to the `ARCHITECTURE` and `TUNE` compiler information in *z/OS C/C++ User's Guide*.
Built-in Functions for Floating-Point and Other Hardware Instructions	Use built-in functions for floating-point and other hardware instructions that are otherwise inaccessible to C/C++ programs. See the appendix on built-in functions in *z/OS C/C++ Programming Guide*.

Chapter 3. z/OS C/C++ Compiler Return Codes and Messages

This chapter contains information about the compiler messages and should not be used as programming interface information.

Return Codes

For every compilation job or job step, the compiler generates a return code that indicates to the operating system the degree of success or failure it achieved:

Table 3. Return Codes from Compilation of a z/OS C/C++ Program

Return Code	Type of Error Detected	Compilation Result
0	No error detected; informational messages may have been issued.	Compilation completed. Successful execution anticipated.
4	Warning error detected.	Compilation completed. Execution may not be successful.
8	Error detected.	Compilation may have been completed. Successful execution not possible.
12	Severe error detected.	Compilation may have been completed. Successful execution not possible.
16	Terminating error detected.	Compilation terminated abnormally. Successful execution not possible.
33	A library level prior to z/OS Language Environment V1R2 was used.	Compilation terminated abnormally. Successful execution not possible.

The return code indicates the highest possible error severity that the compiler detected. Therefore, a particular entry under the ***Types of Error*** column includes **all** error types above it. For example, return code 12 indicates that the compiler has issued a Severe Error and may have also issued any combination of Error, Warning, and Informational messages. But it does not necessarily mean that all these error types are present in that particular compile.

Compiler Messages

Message Format: **CCNnnnn text <&*n*>** where:

nnnn	error message number
text	message which appears on the screen
&*n*	compiler substitution variable

CCN0008 Source file &1 cannot be opened.

Where: &1 is a file name, enclosed in quotes or angle brackets as specified in the corresponding ″include″ directive.

Explanation: The compiler could not open the specified source file.

User Response: Ensure the source file name is correct. Ensure that the correct file is being read and has not been corrupted. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN0015 The compiler could not open the output file ″&1″.

Where: &1 is a file name.

User Response: Ensure the output file name is correct. Also, ensure that the location of the output file has sufficient storage available. If using a LAN drive, ensure that the LAN is working properly and you have permission to write to the disk.

CCN0049 The option ″&1″ is not supported.

Where: &1 is an option

Explanation: The command line contained an option that is not supported. Note that some option parameters must not have spaces between the option and the parameter.

User Response: Remove the option. Check the syntax of the options.

CCN0358 The ″&1″ option is not allowed with the ″&2″ option.

Where: &1 and &2 are both option names.

Explanation: The specified options cannot be used together. The first option specified in the message is ignored.

User Response: Remove one of the options.

CCN0459 An incomplete compile option for ″&1″ has been specified. ″&2″ was expected.

Where: &1 is the option name. &2 is the token that was missing

Explanation: The command line contained an incomplete option. The message identifies what the compiler expected and what it actually found.

User Response: Complete the compile option.

CCN0460 Negative form of option ″&1″ is not allowed.

Where: &1 is the option name.

User Response: Remove the option or change it to the positive form

CCN0461 ″&1″ is not a valid sub-option for ″&2″. Option is ignored.

Where: &1 is the option name.

Explanation: The command line contained an option with an invalid sub-option.

User Response: Remove the sub-option.

CCN0462 ″&1″ must have a sub-option specified.

Where: &1 is the option name.

Explanation: The command line contained an option that was missing a suboption.

User Response: Specify a sub-option.

CCN0463 Sub-option is not allowed in ″&1″ option.

Where: &1 is the option name.

User Response: Remove the sub-option.

CCN0464 ″&1″ requires exactly ″&2″ sub-option(s) to be specified. ″&3″ were given.

Where: &1 is the option name. &2 is the number of options expected.

Explanation: The command line contained an option that had an incorrect number of sub-options specified. The message identifies the number of sub-options the compiler expected and the number it actually found.

User Response: Ensure the correct number of sub-option(s) are given.

CCN0465 ″&1″ requires at most ″&2″ sub-option(s) to be specified. ″&3″ were given.

Where: &1 is the option name. &2 is the number of options expected.

Explanation: The command line contained an option that more sub-options than is allowed for this options. The message identifies the most number of sub-options the compiler expected and the number it actually found.

User Response: Ensure the maximum number of sub-options is not exceeded.

CCN0466 ″&1″ requires at least ″&2″ sub-option(s) to be specified. ″&3″ were given.

Where: &1 is the option name. &2 is the number of options expected.

Explanation: The command line contained an option that fewer sub-options than is allowed for this options. The message identifies the least number of sub-options the compiler expected and the number it actually found.

User Response: Ensure the minimum number of sub-options are specified.

CCN0569 Option ″&1″ is not supported for &2.

Explanation: The option is not supported by this compiler.

User Response: Remove the option.

CCN0611 Unable to access options file &1.

Where: &1 is the options file name specified on OPTFILE option.

Explanation: The compiler could not access the specified options file. It was either unable to open it or unable to read it.

User Response: Ensure the options file name and other specifications are correct. Ensure that the access authority is sufficient. Ensure that the file being accessed has not been corrupted.

CCN0612 Option &1 specified in an options file is ignored.

Where: &1 is an option name specified in the options file.

Explanation: Option &1 is not allowed in an options file.

User Response: Remove the &1 option from the options file. Option OPTFILE can not be nested.

CCN0613 The continuation character on the last line of the options file &1 is ignored.

Explanation: The continuation character on the last line of a file is useless.

User Response: Remove the continuation character on the last line of the options file. Make sure that it is not a typo for something else.

CCN0614 Macro name ″&1″ contains characters not valid on the ″&2″ option.

Explanation: Macro names can contain only alphanumeric characters and the underscore character and must not begin with a numeric character.

User Response: Change the macro name.

CCN0615 Semantic function for processing ″&1″ option is missing.

Explanation: Option &1 cannot be processed because its semantic function is missing.

User Response: Provide the option semantic function.

CCN0623 Option ″&1″ ignored because option ″&2″ specified.

Explanation: Specifying the second option indicated means the first has no effect.

User Response: Remove one of the options.

CCN0624 &1 is not a valid dataset name.

Explanation: The dataset name is not valid because it is too long.

User Response: Use a shorter dataset name.

CCN0625 &1 does not exist.

Explanation: The dataset does not exist.

User Response: Supply an existing dataset.

CCN0626 There are no members in &1 to compile.

Explanation: There are no members in the partitioned dataset to compile.

User Response: Supply a partitioned dataset that contains members.

CCN0627 &1 should be a partitioned dataset.

Explanation: A partitioned dataset is expected.

User Response: Supply a partitioned dataset.

CCN0628 &1 should not be a partitioned dataset.

Explanation: A non-partitioned dataset is expected.

User Response: Supply a non-partitioned dataset.

CCN0629 &1 has invalid attributes.

Explanation: The attributes of the dataset do not match the attributes expected by the compiler.

User Response: Check the informational messages issued with this message and change the dataset attributes accordingly.

CCN0630 &1 has attributes &2.

Explanation: The dataset has the attributes indicated.

User Response: None.

CCN0631 The attributes should be &1.

Explanation: The dataset should have the attributes indicated.

User Response: None.

CCN0632 The attributes should be one of the following:

Explanation: The dataset should have one of the sets of attributes indicated.

User Response: None.

CCN0633 Unable to allocate &1.

Explanation: Unable to allocate the dataset.

User Response: Check that the dataset has a valid name and can be accessed.

CCN0634 Unable to load &1. Compilation terminated.

Explanation: Unable to fetch one of the compiler phases.

User Response: Check that the compiler is installed correctly. Make sure there is enough memory in the region to fetch the module. You may need to specify the runtime option HEAP(,,,FREE,,) to prevent the compiler from running out of memory.

CCN0635 Timestamp error on &1.

Explanation: Timestamp error while compiling a partitioned dataset.

User Response: Check to see if the dataset is corrupted.

CCN0702 An error was encountered in accessing the alternate ddname table. The default ddnames will be used.

Explanation: The compiler could not access the alternate ddname table. Compilation will continue, using the default ddname table.

User Response: Check that the alternate ddname table was coded correctly.

CCN0703 An error was encountered in a call to &1 while processing &2.

Where: &1 is the name of the library function. &2 is the name of the file or path.

Explanation: A library function called by the compiler encountered an error. The compiler will issue a perror() message with more specific information on the failure.

User Response: If the file was created by the user, verify that it was created correctly; See the programmer response for the accompanying perror() message for additional information.

CCN0704 There are no files with the default extension in &1.

Where: &1 is a directory name.

Explanation: There are no files in the given directory which match the default extension. The compiler returned without compiling any files.

User Response: Supply a directory which contains files with the appropriate extension. The default extension for C is ″.c″ and the default extension for C++ is ″.C″.

CCN0705 The output file &1 is not supported in combination with source file &2.

Where: &1 is an output file specified in a compiler option, and &2 is the source file to be compiled.

Explanation: The output file specified in a compiler option is of a type which is not supported in combination with the type of the source file. An informational message describing supported output file types for the given source file type follows.

User Response: Supply an output file of one of the supported types in the compiler sub-option, or let the compiler generate a default output file name.

CCN0706 The source file is a CMS file. The suboption should specify a CMS file or a BFS file in an existing directory.

CCN0707 The source file is a BFS file. The suboption should specify a CMS file, a BFS file in an existing directory, or an existing BFS directory.

CCN0708 The source file is a BFS directory. The suboption should specify an existing BFS directory.

CCN0709 The source file is a Sequential data set. The suboption should specify a sequential data set, a PDS member, or an HFS file in an existing directory.

CCN0710 The source file is a PDS member. The suboption should specify a sequential data set, a PDS member, a PDS, an HFS file in an existing directory, or an existing HFS directory.

CCN0711 The source file is a PDS. The suboption should specify a PDS or an existing HFS directory.

CCN0712 The source file is a HFS file. The suboption should specify a sequential data set, a PDS member, an HFS file in an existing directory, or an existing HFS directory.

CCN0713 The source file is a HFS directory. The suboption should specify an existing HFS directory.

CCN0721 Option ″&1″ cannot be specified with option ″&2″. Option ″&3″ is ignored.

Where: &1 option name, &2 option name, &3 option name.

Explanation: A SEARCH or LSEARCH option cannot be specified on the same compiler invocation with a SYSPATH or USERPATH option. All previous specifications of the conflicting options are ignored.

User Response: Use the correct syntax for specifying the option

CCN0745 &1 should be a partitioned dataset or HFS directory.

Explanation: A partitioned dataset or HFS directory is expected.

User Response: Supply a partitioned dataset or HFS directory.

CCN0750 Suboptions ″&1″ and ″&2″ of option ″&3″ conflict.

Where: &3 is the option name. &1 and &2 are the sub-option names.

User Response: Change the sub-option.

CCN0764 Compiler cannot create temporary files.

Explanation: The intermediate code files could not be created. Please verify that the target file system exists, is writable and is not full.

User Response: Ensure that the designated location for temporary objects exists, is writable and is not full.

CCN0767 The ″&1″ feature of z/OS is not enabled. Contact your system programmer.

Explanation: This feature of z/OS is not enabled at your installation. Your system programmer can contact IBM z/OS service to have this element enabled.

CCN0768 Compiling ″&1″.

Explanation: Informational message issued during PDS or HFS directory compiles to indicate when the compiler has started compiling the next member.

CCN0770 The name &1 is invalid. Please correct and recompile.

Explanation: The name shown is invalid. Please correct the name and retry.

CCN0791 Options ″&1″ and ″&2″ are not compatible.

Where: &1 and &2 are both option names.

Explanation: The specified options cannot be used together.

User Response: Change option values.

CCN0793 Compilation failed for file &1. Object file not created.

Where: &1 is a file name

Explanation: The compiler detected an error and terminated the compilation. Object file was not created.

User Response: Correct the reported errors and recompile.

CCN0795 Unable to open existing dataset &1.

Where: &1 is a dataset name.

Explanation: Although the dataset exists, the compiler was unable to open and/or obtain file information about it.

User Response: Check the informational messages issued with this message and correct the corresponding problems associated with the dataset.

CCN0796 This compiler requires a runtime environment __librel() value of &1.

Where: &1 is the required runtime level in the __librel() format.

Explanation: The compiler cannot run with the current runtime environment because it needs the runtime release indicated.

User Response: Check the informational message issued with this message to determine your current runtime release. Make sure you are running with the runtime environment required.

CCN0797 You are currently running with the runtime environment &1.

Where: &1 is the current runtime level in the __librel() format.

Explanation: The message displays the current runtime level installed on your system.

User Response: None.

CCN0822 Option &1 is locked and cannot be changed.

Explanation: The option has been locked during system installation. The option settings cannot be changed.

User Response: Remove the option from the command line, or ask the system programmer to unlock the option.

CCN0823 Lock suboption &1 is not supported.

Explanation: The lock suboption specified is not supported and is ignored.

User Response: The suboption to the lock option must itself be a valid option. The lock option is set during compiler installation. Check with the system programmer.

CCN1001 INTERNAL COMPILER ERROR: &1.

Explanation: An internal compiler error occurred during compilation.

User Response: Contact your Service Representative.

CCN1002 Virtual storage exceeded.

Explanation: The compiler ran out of memory trying to compile the file. This sometimes happens with large files or programs with large functions. Note that very large programs limit the amount of optimization that can be done.

User Response: Shut down any large processes that are running, ensure your swap path is large enough, turn off optimization, and redefine your virtual storage to a larger size. You can also divide the file into several small sections or shorten the function.

CCN1003 &1.

Where: &1 is the detailed message text.

Explanation: General error message.

CCN1031 Unable to open file ″&1″.

Where: &1 is a file name.

Explanation: The compiler could not open the specified file.

User Response: Ensure the file name is correct. Ensure that the correct file is specified. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN1032 An error occurred while reading file ″&1″.

Where: &1 is a file name.

Explanation: The compiler detected an error while reading from the specified file.

User Response: Ensure that the correct file is being read and has not been damaged. If the file is located on a LAN drive, ensure the LAN is working properly.

CCN1033 An error occurred while writing to file ″&1″.

Where: &1 is a file name.

Explanation: The compiler detected an error while writing to the specified file.

User Response: Ensure that the correct file is specified. If the file is located on a LAN drive, ensure the LAN is working properly.

CCN1034 Read-only pointer initialization of dynamically allocated object &1 is not valid.

Explanation: The value of a read-only pointer must be known at compile time; a pointer cannot be read-only and point to a dynamically allocated object at the same time because the address of the pointee is known at run time only.

User Response: Modify the code so that the pointer is initialized with a read-only value or make the pointer read-write.

CCN1051 Function &1 exceeds size limit.

Explanation: The ACU for the function exceeds the LIMIT specified in the INLINE suboption.

User Response: Increase LIMIT if feasible to do so.

CCN1052 Function &1 is (or grows) too large to be inlined.

Explanation: A function is too large to be inlined into another function.

User Response: Use #pragma inline if feasible to do so.

CCN1053 Some calls to function &1 cannot be inlined.

Explanation: At least one call is either directly recursive, or the wrong number of parameters were specified.

User Response: Check all calls to the function specified and make that number of parameters match the function definition.

CCN1054 Automatic storage for function &1 increased to over &2.

Explanation: The size of automatic storage for function increased by at least 4 KB due to inlining.

User Response: Avoid inlining of functions which have large automatic storage.

CCN1055 Parameter area overflow while compiling &1. Parameter area size exceeds the allowable limit of &2.

Explanation: The parameter area for a function resides in the first 4K of automatic storage for that function. This message indicates that the parameter area cannot fit into 4K.

User Response: Reduce the size of the parameter area by passing fewer parameters or by passing the address of a large structure rather than the structure itself.

CCN1057 &1 section size cannot exceed 16777215 bytes. Total section size is &2 bytes.

Explanation: A Data or Code section cannot exceed 16M in size.

User Response: Partition input source files into multiple source files which can be compiled separately.

CCN1101 Maximum spill size of &2 is exceeded in function &1.

Explanation: Spill size is the size of the spill area. Spill area is the storage allocated if the number of machine registers is not sufficient for program translation.

User Response: Reduce the complexity of the program and recompile.

CCN1102 Spill size for function &1 is not sufficient. Recompile specifying option SPILL(n) where &2 < n <= &3.

Explanation: Spill size is the size of the spill area. Spill area is the storage allocated if the number of machine registers is not sufficient for program translation.

User Response: Recompile using the SPILL(n) option &2 < n <= &3 or with a different OPT level.

CCN1103 Internal error while compiling function &1. &2.

Explanation: An internal compiler error occurred during compilation.

User Response: Contact your Service Representative or compile with a different OPT level.

CCN1104 Internal error while compiling function &1. &2. Compilation terminated.

Explanation: An internal compiler error of high severity has occurred.

User Response: Contact your Service Representative. Be prepared to quote the text of this message.

CCN1105 Constant table overflow compiling function &1. Compilation terminated.

Explanation: The constant table is the table that stores all the integer and floating point constants.

User Response: Reduce the number of constants in the program and recompile.

CCN1106 Instruction in function &1 on line &2 is too complex. Compilation terminated.

Explanation: The specified instruction is too complex to be optimized.

User Response: Reduce the complexity of the instruction and recompile, or recompile with a different OPT level.

CCN1107 Program too complex in function &1.

Explanation: The specified function is too complex to be optimized.

User Response: Reduce the complexity of the program and recompile, or recompile with a different OPT level.

CCN1108 Expression too complex in function &1. Some optimizations not performed.

Explanation: The specified expression is too complex to be optimized.

User Response: Reduce the complexity of the expression or compile with a different OPT level.

CCN1109 Infinite loop detected in function &1. Program may not stop.

Explanation: An infinite loop has been detected in the given function.

User Response: Recode the loop so that it will end.

CCN1110 Loop too complex in function &1. Some optimizations not performed.

Explanation: The specified loop is too complex to be optimized.

User Response: No action is required.

CCN1111 Division by zero detected in function &1. Runtime exception may occur.

Explanation: A division by zero has been detected in the given function.

User Response: Recode the expression to eliminate the divide by zero.

CCN1112 Exponent is non-positive with zero as base in function &1. Runtime exception may occur.

Explanation: This is a possible floating-point divide by zero.

User Response: Recode the expression to eliminate the divide by zero.

CCN1113 Unsigned division by zero detected in function &1. Runtime exception may occur.

Explanation: A division by zero has been detected in the given function.

User Response: Recode the expression to eliminate the divide by zero.

CCN1114 Internal error while compiling function &1. &2.

Explanation: An internal compiler error of low severity has occurred.

User Response: Contact your Service Representative or compile with a different OPT level.

CCN1115 Control flow too complex in function &1; number of basic blocks or edges exceeds &2.

Explanation: Basic blocks are segments of executable code without control flow. Edges are the possible paths of control flow between basic blocks.

User Response: Reduce the complexity of the program and recompile.

CCN1116 Too many expressions in function &1; number of symbolic registers exceeds &2.

Explanation: Symbolic registers are the internal representation of the results of computations.

User Response: Reduce the complexity of the program and recompile.

CCN1117 Too many expressions in function &1; number of computation table entries exceeds &2.

Explanation: The computation table contains all instructions generated in the translation of a program.

User Response: Reduce the complexity of the program and recompile.

CCN1118 Too many instructions in function &1; number of procedure list entries exceeds &2.

Explanation: The procedure list is the list of all instructions generated by the translation of each subprogram.

User Response: Reduce the complexity of the program and recompile.

CCN1119 Number of labels in function &1 exceeds &2.

Explanation: Labels are used whenever the execution path of the program could change; for example: if statements, switch statements, loops or conditional expressions.

User Response: Reduce the complexity of the program and recompile.

CCN1120 Too many symbols in function &1; number of dictionary entries exceeds &2.

Explanation: Dictionary entries are used for variables, aggregate members, string literals, pointer dereferences, function names and internal compiler symbols.

User Response: Compile the program at a lower level of optimization or simplify the program by reducing the number of variables or expressions.

CCN1121 Program is too complex in function &1. Specify MAXMEM option value greater than &2.

Explanation: Some optimizations not performed.

User Response: Recompile specifying option MAXMEM with the suggested value for additional optimization.

CCN1122 Parameter area overflow while compiling &1. Parameter area size exceeds &2.

Explanation: The parameter area is used to pass parameters when calling functions. Its size depends on the number of reference parameters, the number and size of value parameters, and on the linkage used.

User Response: Reduce the size of the parameter area by passing fewer parameters or by passing the address of a large structure rather than the structure itself.

CCN1123 Spill size for function &1 is exceeded. Recompile specifying option SPILL(n) where &2 < n <= &3 for faster spill code.

Explanation: Spill size is the reserved size of the primary spill area. Spill area is the storage allocated if the number of machine registers is not sufficient for program translation.

User Response: Recompile using the SPILL(n) option &2 < n <= &3 for improved spill code generation.

CCN1130 An error occured while opening file ″&1″.

Where: &1 is a file name

Explanation: The compiler could not open the specified file.

User Response: Ensure the file name is correct. Ensure that the correct file is being opened and has not been damaged. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN1131 An error occured while writing file ″&1″.

Where: &1 is a file name

Explanation: The compiler could not read from the specified file.

User Response: Ensure the file name is correct. Ensure that the correct file is being written to and has not been damaged. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN1132 An error occured while closing file ″&1″.

Where: &1 is a file name

Explanation: The compiler could not write to the specified file.

User Response: Ensure the file name is correct. Ensure that the correct file is being closed and has not been damaged. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN1141 Automatic area for &1 is too large

Explanation: Automatic data resides in the stack; the stack size is limited by the target machine addressabilty.

User Response: Avoid large structures and / or arrays as local variables; try using dynamically allocated data. Alternatively, try to break down the procedure into several smaller procedures.

CCN2000 Option ″&1″ is not recognized.

Where: &1 is the option name

Explanation: An invalid option was specified.

User Response: Correct the spelling of the option.

CCN2001 Suboption ″&1″ of option ″&2″ is not supported.

Where: &2 is the option name. &1 is the suboption name.

Explanation: The invocation option contained an unsupported suboption.

User Response: Change the suboption. Check the syntax of the suboption.

CCN2002 Required parameters for option ″&1″ are not specified.

Where: &1 is the option name

Explanation: This option requires that one or more parameters be specified.

User Response: Specify appropriate parameters for the option. Check the option syntax for details.

CCN2003 Parameter ″&1″ of option ″&2″ is not supported.

Where: &2 is the option name. &1 is the option parameter.

Explanation: The parameter for the specified option has invalid syntax.

User Response: Change the option parameter. Check the syntax of the option parameter.

CCN2004 Option ″&1″ parameter error; ″&2″ is not a digit.

Where: &1 is the option name. &2 is invalid character.

Explanation: A non-numeric character was found in the option parameter.

User Response: Change the option parameter. Check the syntax of the option.

CCN2005 ″&1″ is not a decimal number.

Where: &1 is the invalid character.

Explanation: A non-numeric character was found in the option parameter.

User Response: Change the option parameter. Check the syntax of the option.

CCN2010 ″&1″ requires ″&2″ suboptions to be specified. ″&3″ are specified.

Where: &1 is the option name. &2 is the number of options expected. &3 is the number of options specified.

Explanation: An incorrect number of suboptions was specified for this option. The message identifies the number of suboptions the compiler expected and the number it actually found.

User Response: Ensure the correct number of suboptions are specified.

CCN2011 At most ″&2″ suboptions must be specified for &1. ″&3″ are specified.

Where: &1 is the option name. &2 is the number of options expected. &3 is the number of options specified.

Explanation: Too many suboptions were specified for this option.

User Response: Ensure that the maximum number of suboptions is not exceeded.

CCN2012 ″&1″ requires at least ″&2″ suboptions to be specified. ″&3″ are specified.

Where: &1 is the option name. &2 is the number of options expected. &3 is the number of options specified.

Explanation: Not enough suboptions were specified for this option.

User Response: Ensure that the minimum number of suboptions are specified.

CCN2013 Suboptions ″&1″ and ″&2″ of option ″&3″ conflict.

Where: &3 is the option name. &1 and &2 are the suboption names.

User Response: Determine which suboption is required. Remove the other suboption to eliminate the conflict.

CCN2015 Incompatible specifications for options ARCH and TUNE.

User Response: Determine what target machine/architecture family is desired and select a compatible target machine for tuning.

CCN2020 Option ″&1″ is turned on because option ″&2″ is specified.

Where: &1 and &2 are both option names.

Explanation: If you specify option &2, the compiler turns on option &1 to achieve a better options combination.

User Response: Specify option &1 to eliminate this message.

CCN2021 Option ″&1″ is ignored because option ″&2″ was specified.

Where: &1 and &2 are both option names.

Explanation: Specifying the second option indicated means the first has no effect.

User Response: Remove one of the options.

CCN2022 Option ″&1″ is not supported for IPA processing.

Where: &1 is an option name.

Explanation: The specified option (or corresponding #pragma) is not supported for an IPA compilation. Processing is terminated.

User Response: Correct the option or #pragma specification, as appropriate.

CCN2023 Option ″&1″ has been promoted to ″&2″ because option ″&3″ was specified.

Where: &1, &2 and &3 are all option names.

Explanation: Specifying the &3 option caused sufficient information to be available to support the &2

option instead of the &1 option.

User Response: None

CCN2030 &1

Where: &1 is the detailed message text.

Explanation: General informational message.

CCN2031 &1

Where: &1 is the detailed message text.

Explanation: General warning message.

CCN2032 &1

Where: &1 is the detailed message text.

Explanation: General error message.

CCN2033 &1

Where: &1 is the detailed message text.

Explanation: General severe error message.

CCN2050 IPA Link control file: Syntax error.

Explanation: A syntax error was detected in the IPA Link control file. Processing is terminated.

User Response: Correct the IPA Link control file syntax.

CCN2051 IPA Link control file: Unmatched quote.

Explanation: A quoted string representing a directive operand was detected in the IPA Link control file, but this string was not terminated by a matching quote before the end of file. Processing is terminated.

User Response: Correct the IPA Link control file operand syntax.

CCN2052 IPA Link control file: Directive ″&1″ is incorrect.

Where: &1 is the directive in error.

Explanation: An incorrectly specified directive was detected in the IPA Link control file. The directive is ignored, and processing continues.

User Response: Correct the specified directive in the IPA Link control file.

CCN2053 IPA Link control file: &1.

Where: &1 is the detailed message text.

Explanation: An error was detected in the IPA Link control file. Processing is terminated.

User Response: Correct the specified IPA Link control file error.

CCN2059 IPA Link control file: INTERNAL COMPILER ERROR - &1.

Where: &1 is the detailed message text.

Explanation: An internal compiler error occurred during processing of the IPA Link control file.

User Response: Contact your Service Representative and provide the detailed message text.

CCN2060 CSECT name entry &1 (″&2″) is not unique. It conflicts with entry &3.

Where: &1 and &3 are CSECT name entry numbers, &2 is the CSECT name entry.

Explanation: The specified CSECT name prefix entry in the IPA Link control file duplicates an previous CSECT name prefix entry.

User Response: Provide a unique value for the CSECT name prefix that caused the conflict.

CCN2061 A CSECT name prefix is not specified for partition &1. The CSECT option is active.

Where: &1 is the number of the current partition.

Explanation: The CSECT option is active, which requires that a CSECT name prefix entry be specified in the IPA Link control file for each partition in the generated object module. A system-generated name prefix has been provided for the current partition.

User Response: Provide one or more additional CSECT name prefixes so that each partition will have a unique name.

CCN2062 A CSECT name prefix is not specified for partition &1.

Where: &1 is the number of the current partition.

Explanation: One or more CSECT name prefixes were specified in the IPA Link control file, but there were insufficient entries for all partitions in the generated object module. The CSECT option is not active, so these missing names are not considered an error. A system-generated name prefix has been provided for the current partition.

User Response: Provide one or more additional CSECT name prefixes so that each partition will have a unique name.

CCN2100 No object files were specified as input to the IPA Link step.

Explanation: No object files were specified for IPA Link step processing.

User Response: Specify at least one object file.

CCN2101 No IPA object was found.

Explanation: IPA object information was not found during IPA Link step processing.

User Response: Ensure that the appropriate object files include IPA object information.

CCN2102 IPA object information is missing ″&1″ records.

Where: &1 is an object record type.

Explanation: A damaged IPA object file was encountered during IPA Link step processing.

User Response: Recompile the source file and retry IPA Link step processing. If the problem persists, call your Service Representative.

CCN2103 IPA object information has invalid ″&1″ record.

Where: &1 is an object record type.

Explanation: A damaged IPA object file was encountered during IPA Link step processing.

User Response: Recompile the source file and retry IPA Link step processing. If the problem persists, call your Service Representative.

CCN2104 Object information is missing ″&1″ records.

Where: &1 is an object record type.

Explanation: A damaged non-IPA object file was encountered during IPA Link step processing.

User Response: Recompile the source file and retry IPA Link step processing. If the problem persists, call your Service Representative.

CCN2105 Object information has an invalid ″&1″ record.

Where: &1 is an object record type.

Explanation: A damaged non-IPA object file was encountered during IPA Link step processing.

User Response: Recompile the source file and retry IPA Link step processing. If the problem persists, call your Service Representative.

CCN2106 An error was encountered during object information processing.

Where: &1 is an object record type.

Explanation: A damaged or incompatible object file was encountered during IPA Link step processing.

User Response: Recompile the source file and retry IPA Link step processing. If the problem persists, call your Service Representative.

CCN2107 ″&1″ is not the first symbol on the object record.

Where: &1 is an object record type.

Explanation: A damaged IPA object file was encountered during IPA Link step processing.

User Response: Recompile the source file and retry IPA Link step processing. If the problem persists, call your Service Representative.

CCN2108 Object information has incorrect format.

Explanation: An object file with an incorrect format was encountered during IPA Link step processing.

User Response: Recompile the source file and retry IPA Link step processing. If the problem persists, call your Service Representative.

CCN2109 Generated file is too big. Reduce partition size or turn off IPA.

Explanation: The file generated by IPA exceeds encoding limits.

User Response: Relink with a reduced partition size or without IPA.

CCN2110 ″&1″ IPA Link control statement has no specifications.

Where: &1 is either INCLUDE, LIBRARY, AUTOCALL, IMPORT or ENTRY.

Explanation: An IPA Link control statement object record without any specifications was encountered during processing. The record is ignored. Processing continues.

User Response: If the IPA Link control statement is required, provide appropriate INCLUDE, LIBRARY, or AUTOCALL, IMPORT or ENTRY specifications and repeat the step. If the record is not required, the warning message can be removed by deleting the invalid record.

CCN2111 Invalid syntax specified on ″&1″ IPA Link control statement.

Where: &1 is either INCLUDE, LIBRARY, AUTOCALL, IMPORT, ENTRY, or UNKNOWN.

Explanation: An IPA Link control statement object record with invalid syntax was encountered during processing. The record is processed up to the syntax error and the remainder of the record is ignored. Processing continues. If unmatched quotes were encountered, the IPA LINK control statement type will be listed as ″UNKNOWN″.

User Response: If the IPA Link control statement is required, correct the syntax errors and repeat the step. If the record is not required, the warning message can be removed by deleting the invalid record.

CCN2112 Continuation record missing for ″&1″ IPA Link control statement.

Where: &1 is the IPA Link control statement type.

Explanation: An IPA Link control statement object record of type &1 was encountered with the continuation column set, but there was no subsequent record or the subsequent record was not a valid continuation record. The record is ignored and processing continues.

User Response: Add the appropriate continuation record, or set continuation column 72 to blank if no continuation record is required.

CCN2113 Continuation records not allowed for ″&1″ IPA Link control statement. This statement was ignored.

Where: &1 is the IPA Link control statement type.

Explanation: An IPA Link control statement of type &1 had a nonblank character in column 72. Information for a statement of this type must be specified in one record, so continuation of this record is not valid. The statement is ignored and IPA Link step processing continues.

User Response: Correct the record if necessary, set continuation column 72 to blank, and repeat the step.

CCN2114 More than one ″&1″ IPA Link control statement found.

Where: &1 is the IPA Link control statement type.

Explanation: More than one IPA Link control statement object record of type &1 was encountered during the processing of &2.

User Response: No recovery is necessary unless the incorrect IPA Link control statement is selected by IPA Link error recovery, or incorrect processing was performed. In this case, remove the offending record and repeat the step.

CCN2115 ″&1″ IPA Link control statement is ignored.

Where: &1 is the control statement type.

Explanation: An IPA Link control statement of type &1 was found to be invalid. The record is ignored and processing continues.

User Response: Correct the record if necessary, set continuation column 72 to blank, and repeat the step.

CCN2116 An error occurred processing the ″&1″ IPA Link control statement.

Where: &1 is either INCLUDE, LIBRARY, AUTOCALL, IMPORT or ENTRY.

Explanation: An error was encountered during processing of the IPA Link control statement. The record is ignored and processing continues.

User Response: Ensure that the files referenced by this IPA Link control statement object record are available and in the correct format. If the problem persists, call your Service Representative.

CCN2117 ″&1″ IPA Link control statement specification not supported.

Where: &1 is either INCLUDE, LIBRARY, AUTOCALL, IMPORT or ENTRY.

Explanation: An IPA Link control statement with a specification syntax that is unsupported by IPA Link was encountered during processing. The record is processed up to this specification, and the remainder of the record is ignored. Processing continues.

User Response: Alter the specification to a format supported by IPA Link, or remove the specification. If the record is not required, the warning message can be removed by deleting the invalid record.

CCN2119 Noobject files used in non-IPA link step.

Explanation: One or more files generated with ″NOOBJECT″ were being linked directly by the linker.

User Response: Recompile and link with ″OBJECT″ or recompile the file containing the entry point with IPA.

CCN2120 IPA Link control statement has invalid syntax:

Explanation: An IPA Link control statement object record (related to DLL resolution) with invalid syntax was encountered during processing.

User Response: Prelink the DLL and generate a valid definition side-deck file.

CCN2121 IPA Link control statement not properly continued:

Explanation: An IPA Link control statement object record (related to DLL resolution) with the continuation column set was encountered, but there was no subsequent record or the subsequent record was not a valid continuation record. The record is ignored and processing continues.

User Response: Prelink the DLL and generate a valid definition side-deck file.

CCN2122 Module name ″&1″ chosen for generated ″IMPORT″ IPA Link control statements.

Where: &1 is a module name.

Explanation: The default name TEMPNAME was assigned to the module in the DLL definition side-deck file.

User Response: Provide a ″NAME″ IPA Link control statement.

CCN2125 File ″&1″ is sequential format. The member name ″&2″ can not be specified on the ″&3″ IPA Link control statement.

Where: &1 is a file name. &2 is a member name. &3 is INCLUDE.

Explanation: An IPA Link control statement specification is syntactically correct, but is incorrect for the sequential file which has been allocated. This specification is ignored, and processing continues.

User Response: Ensure the file allocation specification is correct. Correct the file allocation or IPA Link control statement as necessary and repeat the step.

CCN2126 File ″&1″ is partitioned format. A member name must be specified on the ″&2″ IPA Link control statement.

Where: &1 is a file name. &2 is INCLUDE.

Explanation: An IPA Link control statement specification is syntactically correct, but is incorrect for the partitioned file which has been allocated. This specification is ignored, and processing continues.

User Response: Ensure the file allocation specification is correct. Correct the file allocation or IPA Link control statement as necessary and repeat the step.

CCN2127 File ″&1″ is sequential format. A partitioned file or OE archive is required for a ″&2″ IPA Link control statement.

Where: &1 is a file name. &2 is LIBRARY.

Explanation: An IPA Link control statement specification is syntactically correct, but the corresponding file is sequential format. This specification is ignored, and processing continues.

User Response: Ensure the file allocation specification is correct. Correct the file allocation as necessary and repeat the step.

CCN2128 File ″&1″ is sequential format. A partitioned file or OE archive is required for Autocall processing.

Where: &1 is a file name.

Explanation: The specified file is allocated to a sequential file, and is unavailable for autocall processing.

User Response: Ensure the file allocation specification is correct. Correct the file allocation as necessary and repeat the step.

CCN2130 A ″RENAME″ IPA Link control statement can not be used for short name ″&1″.

Where: &1 is a short name.

Explanation: A ″RENAME″ IPA Link control statement object record that attempted to rename a short name &1 to another name was encountered. ″RENAME″ statements are only valid for long names for which there are no corresponding short names. The ″RENAME″ statement is ignored and processing continues.

User Response: The warning message can be removed by deleting the invalid ″RENAME″ statement.

CCN2131 Multiple ″RENAME″ IPA Link control statements are found for ″&1″. The first valid one is used.

Where: &1 is a name.

Explanation: More than one ″RENAME″ IPA Link control statement object record was encountered for name &1. The first ″RENAME″ statement with a valid output name is chosen. The ″RENAME″ statement is ignored and processing continues.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the “Object File Map” section of the listing to determine which output name was chosen. If it was not the intended name, remove the duplicate ″RENAME″ statements and repeat the step.

CCN2132 May not ″RENAME″ long name ″&1″ to another long name ″&2″.

Where: &1 and &2 are both long names.

Explanation: A ″RENAME″ IPA Link control statement object record that attempted to rename a long name &1 to another long name &2 was encountered. The ″RENAME″ statement is ignored and processing continues.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the "Object File Map" section of the listing to determine which output name was chosen. If it was not the intended name, replace the invalid ″RENAME″ statement with a valid output name and repeat the step. The warning message can be removed by deleting the invalid RENAME statement.

CCN2133 May not ″RENAME″ defined long name ″&1″ to defined name ″&2″.

Where: &1 is a long name. &2 is a defined name.

Explanation: A ″RENAME″ IPA Link control statement object record that attempted to rename a defined long name &1 to another defined name &2 was encountered. The ″RENAME″ statement is ignored and processing continues.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the "Object File Map" section of the listing to determine which output name was chosen. If it was not the intended name, replace the invalid ″RENAME″ statement with a valid output name and repeat the step. The warning message can be removed by deleting the invalid RENAME statement.

CCN2134 ″RENAME″ of ″&1″ to ″&2″ is ignored since ″&2″ is the target of another ″RENAME″.

Where: &1 is a long name. &2 is a defined name.

Explanation: Multiple ″RENAME″ IPA Link control statement object records that attempted to rename two different names to the same name &2 were encountered. The ″RENAME″ statement is ignored and processing continues.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the "Object File Map" section of the listing to determine which name was renamed to &2. If it was not the intended name, change the name and repeat the step. The warning message can be removed by deleting the extra ″RENAME″ statements.

CCN2140 ″&1″ is mapped to ″&2″ by the IPA(UPCASE) option. ″&3″ is an alternative matching definition name.

Where: &1, &2 and &3 are names.

Explanation: ″&1″ is an external symbol reference that maps to multiple definitions due to the IPA(UPCASE) option. Definition ″&2″ was selected. ″&3″ is another definition which matches this name, but was not used.

User Response: If both names (&1 and &2) correspond to the same object the warning can be ignored. If the names do not correspond to the same object or if the warning is to be removed, do one of the following:

- Change one of the names in the source routine.
- Use #pragma map in the source routine for one of the names.

CCN2141 ″&1″ is mapped to ″&2″.

Where: &1 and &2 are names.

Explanation: External name ″&1″ has been replaced by ″&2″. IPA Link processing required a name that was limited to 8 characters.

User Response: None. If you require a specific external name for ″&1″, use #pragma map in the program source. Any additional names that were mapped to ″&1″ (and hence ″&2″) because of IPA(UPCASE) will require equivalent #pragma map statements.

CCN2142 Unable to map ″&1″ and ″&2″ to a common name during IPA(UPCASE) processing.

Where: &1 and &2 are names.

Explanation: Due to references by non-IPA objects, a common external name can not be determined during IPA(UPCASE) processing. This will occur if both ″&1″ and ″&2″ are referenced by non-IPA objects, or if either is referenced by non-IPA objects and the common name is longer than 8 characters.

User Response: Modify the program source so that the external names are consistent, and 8 characters or less in length.

CCN2143 Unable to map ″&1″ to ″&2″ within same Compilation Unit during IPA(UPCASE) processing.

Where: &1 and &2 are names.

Explanation: ″&1″ is an external symbol that maps to the symbol ″&2″ within the same Compilation Unit due to the IPA(UPCASE) option. Mapping of symbols in this manner is not supported.

User Response: Modify the program source so that the external names are consistent. If IPA(UPCASE) resolution is desired, split the program source so that each symbol is defined in a different Compilation Unit.

CCN2150 Invalid C370LIB-directory encountered.

Explanation: The specified library file contains an invalid or damaged C370LIB-directory.

User Response: Use the C370LIB DIR command to recreate the C370LIB-directory, and repeat the step.

CCN2151 Library does not contain a C370LIB-directory.

Explanation: The specified library file does not contain a C370LIB-directory required to perform the command.

User Response: The library was not created with the C370LIB command. Use the C370LIB DIR command to create the C370LIB-directory, and repeat the step.

CCN2152 Member ″&1″ not found in library.

Where: &1 is a library member name.

Explanation: The specified member &1 was not found in the library. Processing continues.

User Response: Use the C370LIB MAP command to display the names of library members.

CCN2153 Unable to access library file.

Explanation: An error was encountered during processing of the specified ″LIBRARY″ IPA Link control statement. The record is ignored and processing continues.

User Response: Ensure that the files referenced by this IPA Link control statement object record are available and in the correct format. If the problem persists, call your Service Representative.

CCN2155 &1 sequential files in library ″&2″ allocation were ignored.

Where: &1 is the number of sequential files. &2 is a library DD name.

Explanation: When the list of files allocated to the specified DD was extracted, both sequential and partitioned format files were found. The sequential files were ignored.

User Response: Correct the library allocation to eliminate the sequential files.

CCN2160 Invalid symbol table encountered in archive library.

Explanation: The specified archive library file contains invalid information in its symbol table. Processing continues.

User Response: Rebuild the archive library.

CCN2161 Archive library does not contain a symbol table.

Explanation: The symbol table for the specified archive library file could not be found.

User Response: Rebuild the archive library.

CCN2170 Unresolved ″IMPORT″ references are detected.

Explanation: Unresolved objects were encountered at IPA Link processing termination. Other user objects are required.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the “Object Resolution Warnings” section of the listing to find the objects in question. To correct unresolved references to user objects, include the user objects during IPA Link processing.

CCN2171 Unresolved ″IMPORT″ references are detected:

Explanation: The listed unresolved objects were encountered at IPA Link processing termination. Other user objects are required.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the “Object Resolution Warnings” section of the listing to find the objects in question. To correct unresolved references to user objects, include the user objects during IPA Link processing.

CCN2172 Unresolved references could not be imported.

Explanation: The same symbol was referenced in both DLL and non-DLL code. The DLL reference could have been satisfied by an ″IMPORT″ IPA Link control statement which was processed, but the non-DLL reference could not.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the “Object Resolution Warnings” section of the listing to find the symbols in question. You must either supply a definition for the referenced symbol, or use the DLL compiler option to recompile the code containing the non-DLL reference so that it becomes a DLL reference.

CCN2173 Unresolved references could not be imported:

Explanation: The listed symbols were referenced in both DLL and non-DLL code. The DLL reference could have been satisfied by an ″IMPORT″ IPA Link control statement which was processed, but the non-DLL reference could not.

User Response: You must either supply a definition for the referenced symbol, or use the DLL compiler option to recompile the code containing the non-DLL reference so that it becomes a DLL reference.

CCN2174 Duplicate ″IMPORT″ definitions are detected.

Explanation: A name referenced in DLL code was not defined within the application, but more than one ″IMPORT″ IPA Link control statement was detected with that symbol name. The first one encountered was used.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the “Object Resolution Warnings” section of the listing to find the objects in question, and define these objects once.

CCN2175 Duplicate ″IMPORT″ definitions are detected:

Explanation: The listed objects were defined multiple times.

User Response: Define these objects once.

CCN2177 ″ENTRY″ symbol ″&1″ not found.

Where: &1 is a symbol name.

Explanation: An ″ENTRY″ IPA Link control statement object record that attempted to specify a program entry point was encountered, but no symbol by this name is present in the application program.

User Response: If the IPA Link control statement is required, provide an object file which defines the symbol, and repeat the step. If the record is not required, the error message can be removed by deleting the invalid record.

CCN2178 ″ENTRY″ symbol ″&1″ not valid.

Where: &1 is a symbol name.

Explanation: An ″ENTRY″ IPA Link control statement object record that attempted to specify a program entry point was encountered, but the specified symbol is a reference, or aggregate member.

User Response: If the IPA Link control statement is required, provide an object file which defines a valid symbol, and repeat the step. If the record is not required, the error message can be removed by deleting the invalid record.

CCN2180 Load Module information has invalid ″&1″ record.

Where: &1 is an Load Module record type.

Explanation: A damaged or incompatible Load Module library member was encountered during IPA Link processing.

User Response: Recompile the source file and retry IPA Link processing. If the problem persists, call your Service Representative.

CCN2181 An error was encountered during Load Module information processing.

Where: &1 is an Load Module record type.

Explanation: A damaged or incompatible Load Module library member was encountered during IPA Link processing.

User Response: Recompile the source file and retry IPA Link processing. If the problem persists, call your Service Representative.

CCN2182 Load Module information has incorrect format.

Explanation: A Load Module library member with an incorrect format was encountered during IPA Link processing.

User Response: Recompile the source file and retry IPA Link processing. If the problem persists, call your Service Representative.

CCN2183 Program Object file format is not supported by IPA Link step processing.

Explanation: During the link portion of IPA Link step processing, an attempt was made to extract object information from a Program Object file. IPA Link step processing supports object information in the form of object modules, and Load Module library members. Program Object files which are generated by the Program Management Binder are not supported.

User Response: Repackage the Program Object as either an object module or a Load Module library member, and retry IPA Link processing.

CCN2184 IPA Object file ″&1″ has been compiled with an incompatible version of IPA.

Explanation: The IPA Object format in ″&1″ is incompatible with the current compiler.

User Response: Recompile the file with the current compiler.

CCN2185 The correct decryption key for object file ″&1″ was not specified.

Explanation: The file ″&1″ was encrypted with different key than the one(s) specified.

User Response: Include the correct key or link without IPA.

CCN2200 Unresolved references to writable static objects are detected.

Explanation: Undefined writable static objects were encountered at IPA Link step processing termination. Other user objects are required.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the “Object Resolution Warnings” section of the listing to find the objects in question, and include these objects during IPA Link processing.

CCN2201 Undefined writable static objects are detected:

Explanation: The listed writable static objects were undefined at IPA Link processing termination.

User Response: Include these objects during IPA Link processing.

CCN2202 Unresolved references to writable static objects are detected:

Explanation: Undefined writable static objects or unresolved objects referring to writable static objects were encountered at IPA Link processing termination. Other user objects are required.

User Response: Include these objects during IPA Link processing.

CCN2203 Unresolved references to objects are detected.

Explanation: Unresolved objects were encountered at IPA Link processing termination. Other user objects are required.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the “Object Resolution Warnings” section of the listing to find the objects in question. To correct unresolved references to user objects, include the required objects during IPA Link processing.

CCN2204 Unresolved references to objects are detected:

Explanation: The listed unresolved objects were encountered at IPA Link processing termination. Other user objects are required.

User Response: To correct the unresolved references, include the required objects during IPA Link step processing.

CCN2205 Unresolved reference to symbol ″&1″.

Explanation: The listed unresolved objects were encountered at IPA Link processing termination. Other user objects are required.

User Response: To correct the unresolved references, include the required objects during IPA Link step processing.

CCN2206 Unresolved reference to symbol ″&1″.

Explanation: The listed unresolved objects were encountered at IPA Link processing termination. Other user objects are required.

User Response: To correct the unresolved references, include the required objects during IPA Link step processing.

CCN2210 Duplicate writable static objects are detected.

Explanation: Writable static objects were defined multiple times.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the “Object Resolution Warnings” section of the listing to find the objects in question, and define the required objects once.

CCN2211 Duplicate writable static objects are detected:

Explanation: The listed writable static objects were defined multiple times.

User Response: Define these objects once.

CCN2212 Duplicate objects are detected.

Explanation: Objects were defined multiple times.

User Response: Specify the IPA(LINK,MAP) option during processing. Examine the “Object Resolution Warnings” section of the listing to find the objects in question, and define these objects once.

CCN2213 Duplicate objects are detected:

Explanation: The listed objects were defined multiple times.

User Response: Define the objects once.

CCN2220 Duplicate writable static object ″&1″ is detected with different sizes. The largest size is used.

Where: &1 is a writable static object name.

Explanation: The listed writable static object was defined multiple times with different sizes. The larger of the different sizes was used. Incorrect execution could occur unless the object is defined consistently.

User Response: Define the objects consistently.

CCN2221 Duplicate object ″&1″ is detected with different sizes. The largest size is used.

Where: &1 is an object name.

Explanation: The listed object was defined multiple times with different sizes. The larger of the different sizes is used. Incorrect execution could occur unless the object is defined consistently.

User Response: Define these objects consistently.

CCN2229 No exported symbols found.

Explanation: After the IPA object files were linked, an unsuccessful attempt was made to locate at least one exported symbols.

User Response: Specify at least one exported symbol contained in the IPA object files.

CCN2230 Program entry point not found.

Explanation: After the IPA object files were linked, an unsuccessful attempt was made to identify the program entry point (normally the ″main″ function).

User Response: Provide the IPA object file containing the program entry point.

CCN2231 More than one entry point was found.

Explanation: After the IPA object files were linked, multiple possible program entry points were found.

User Response: Eliminate the IPA object files containing the extra program entry points.

CCN2232 Duplicate definition of symbol ″&1″ ignored.

Where: &1 is the symbol name.

Explanation: A duplicate definition of the specified symbol has been encountered in the specified file. It is ignored.

User Response: If possible, eliminate the duplicate symbol definition from the set of input files provided to the IPA Link step.

CCN2233 Duplicate definition of symbol ″&1″ in import list is ignored.

Where: &1 is the symbol name.

Explanation: A duplicate definition of the specified symbol has been encountered in an import list in the specified file. It is ignored.

User Response: Eliminate the duplicate import definition for the specified symbol.

CCN2240 IPA object files ″&1″ and ″&2″ have been compiled with differing settings for the ″&3″ option.

Where: &1 and &2 are object file names, and &3 is an option name.

Explanation: The IPA object files were compiled using conflicting settings for the specified option. A final common option setting will be selected. Alternatively, a common override can be specified during IPA Link invocation.

User Response: Ensure that the final option setting is appropriate. The warning message can be removed by recompiling one or both source files with the same option setting.

CCN2241 The ″&1″ option will be used.

Where: &1 is an option name.

Explanation: This is the final common option setting selected after IPA object files were found to be in conflict.

User Response: Ensure that the final option setting is appropriate. The warning message can be removed by recompiling one or both source files with the same option setting.

CCN2242 IPA object files ″&1″ and ″&2″ contain code targeted for different machine architectures.

Where: &1 and &2 are object file names.

Explanation: The IPA object files were compiled with conflicting machine architectures. A final common machine architecture will be selected.

User Response: Ensure that the final machine architecture is appropriate. The warning message can be removed by recompiling one or both source files so that consistent ARCH options that specify the same machine architecture are used.

CCN2243 The ″&1″ machine architecture will be used.

Where: &1 is a machine architecture id.

Explanation: This is the final machine architecture selected after IPA object files were found to be in conflict.

User Response: Ensure that the final machine architecture is appropriate. The warning message can be removed by recompiling one or both source files so that consistent ARCH options that specify the same machine architecture are used.

CCN2244 IPA object files ″&1″ and ″&2″ contain code targeted for different operating environments.

Where: &1 and &2 are object file names.

Explanation: The IPA object files were compiled using conflicting operating environments. A final common operating environment will be selected.

User Response: Ensure that the final target operating environment is appropriate. The warning message can be removed by recompiling one or both source files for the same operating environment.

CCN2245 The ″&1″ operating environment will be used.

Where: &1 is an operating environment id.

Explanation: This is the final operating environment selected after IPA object files were found to be in conflict.

User Response: Ensure that the final target operating environment is appropriate. The warning message can be removed by recompiling one or both source files for the same operating environment.

CCN2246 IPA object files ″&1″ and ″&2″ were generated from different source languages.

Where: &1 and &2 are object file names.

Explanation: The IPA object files were produced by compilers for different languages. The IPA object has been transformed as required to handle this situation.

User Response: None.

CCN2247 IPA object files ″&1″ and ″&2″ were generated by different compiler versions.

Where: &1 and &2 are object file names.

Explanation: The IPA object files were produced by different versions of the compiler. The older IPA object has been transformed to the later version.

User Response: None.

CCN2248 The code page for one or more IPA object files differs from the code page ″&1″, used during IPA Link processing.

Where: &1 is a code page name.

Explanation: IPA object files contain code page identification if the LOCALE option is active when they are originally compiled. During IPA Link processing with the LOCALE option active, one or more IPA object files were encountered that had a code page (specified via the LOCALE option) which differs from that used during IPA Link processing. Character data will remain in the code page in which it was originally compiled.

User Response: None.

CCN2250 Option ″&1″ not available because one or more IPA object files were compiled with option ″&2″.

Where: &1 and &2 are option names.

Explanation: The specified option is not available during code generation for the current partition, because one or more IPA object files contain insufficient information to support it. A final common option will be selected.

CCN2260 Subprogram specified exceeds size limit: &1

Where: &1 is the Subprogram name.

Explanation: The ACU for the subprogram exceeds the LIMIT specified in the INLINE suboption.

User Response: Increase LIMIT if it is feasible to do so.

CCN2261 Subprogram specified is (or grows) too large to be inlined: &1

Where: &1 is the subprogram name.

Explanation: This occurs when a subprogram is too large to be inlined into another subprogram.

User Response: Use #pragma inline if it is feasible to do so.

CCN2262 Some calls to subprogram specified cannot be inlined: &1

Where: &1 is the subprogram name.

Explanation: At least one call is either directly recursive, or the wrong number of parameters were specified.

User Response: Check all calls to the subprogram specified and make sure that the number of parameters match the subprogram definition.

CCN2263 Automatic storage for subprogram specified increased to over &1 bytes: &2

Where: &1 is the automatic storage limit. &2 is the subprogram name.

Explanation: The size of automatic storage for subprogram increased by at least 4 KB due to inlining.

User Response: If feasible to do so, prevent the inlining of subprograms that have large auto storage.

CCN2265 Inlining of specified subprogram failed due to the presence of a global label: &1

Where: &1 is the subprogram name.

Explanation: At least one call could not be inlined due to the presence of a global label.

User Response: Minimize the use of global labels in your application. Their presence will inhibit global inlining.

CCN2266 Inlining of specified subprogram failed due to the presence of a C++ exception handler: &1

Where: &1 is the subprogram name.

Explanation: At least one call could not be inlined due to the presence of a C++ exception handler.

User Response: Minimize the use of C++ exception handlers in your application. Their presence will inhibit global inlining.

CCN2267 Inlining of specified subprogram failed due to the presence of variable arguments: &1

Where: &1 is the subprogram name.

Explanation: At least one call could not be inlined due to the presence of variable arguments.

User Response: None.

CCN2268 Inlining of subprogram ″&1″ into subprogram ″&2″ failed due to a conflict in options settings.

Where: &1 and &2 are subprogram names.

Explanation: The specified call could not be inlined due to incompatible options settings for the IPA object files that contain the two programs.

User Response: Use compatible options during the IPA Compile step.

CCN2269 Inlining of subprogram ″&1″ into subprogram ″&2″ failed due to a type mismatch in argument ″&3″.

Where: &1 and &2 are subprogram names. &3 is the parameter index

Explanation: The specified call could not be inlined due to incompatible types for the specified argument number, where ″&1″ is the first argument.

User Response: Correct the program to use compatible types for all arguments.

CCN2270 Subprogram ″&1″ has been inlined into subprogram ″&2″. One or more unexpected extra parameters were ignored.

Where: &1 and &2 are subprogram names.

Explanation: The specified call was inlined, but one or more parameters on the call were not required and were ignored.

User Response: Eliminate the extra parameters.

CCN2271 Subprogram ″&1″ has been inlined into subprogram ″&2″. One or more arguments were not supplied, so the values are undefined.

Where: &1 and &2 are subprogram names.

Explanation: The specified call was inlined, but one or more parameters were omitted on the call. Values for these arguments are indeterminate, so the operation of the subprogram is undefined.

User Response: Specify all parameters actually required by the called subprogram.

CCN2280 A type mismatch was detected for symbol ″&1″.

Where: &1 is a subprogram name.

Explanation: An instance of the specified subprogram was found where one or more parameters were of an unexpected type.

User Response: Correct the program to use parameter types compatible with the function definition. .

CCN2281 Function return types ″&1″ and ″&2″ for subprogram ″&3″ do not match.

Where: &1 and &2 are return type names. &3 is a subprogram name.

Explanation: An instance of the specified subprogram was found with an unexpected type for the function return value.

User Response: Correct the program to use a return type compatible with the function definition.

CCN2282 Subprogram ″&1″ has the wrong number of formal parameters.

Where: &1 is a subprogram name.

Explanation: The number of formal parameters for the definition of the given subprogram does not match the number of formal parameters for the declaration of the subprogram.

User Response: Correct the program to use a consistent number of formal parameters for the subprogram.

CCN2283 A linkage mismatch was detected for symbol ″&1″.

Where: &1 is a symbol name.

Explanation: An instance of the specified subprogram was found which uses a linkage incompatible with the calling function.

User Response: Correct the program to ensure consistent linkage across all objects.

CCN2299 Some optimizations may be inhibited.

Explanation: During optimization of the IPA object, a problem was encountered that prevent the use of all available optimization techniques. These specific problems are identified in separate messages.

User Response: Correct the problem which inhibits optimization.

CCN2300 Export symbol ″&1″ not found.

Where: &1 is a symbol name.

Explanation: An ″export″ directive entry for the specified symbol was present in the IPA Link control file, but no symbol by this name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2301 External subprogram ″&1″ not found. Could not mark as ″pure″.

Where: &1 is a subprogram name.

Explanation: A ″pure″ directive entry for the specified subprogram was present in the IPA Link control file, but no subprogram by this name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2302 External subprogram ″&1″ not found. Could not mark as ″isolated″.

Where: &1 is a subprogram name.

Explanation: A ″isolated″ directive entry for the specified subprogram was present in the IPA Link control file, but no subprogram by this name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2303 External subprogram ″&1″ not found. Could not mark as ″safe″.

Where: &1 is a subprogram name.

Explanation: A ″safe″ directive entry for the specified subprogram was present in the IPA Link control file, but no subprogram by this name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2304 External subprogram ″&1″ not found. Could not mark as ″unknown″.

Where: &1 is a subprogram name.

Explanation: An ″unknown″ directive entry for the specified subprogram was present in the IPA Link control file, but no subprogram by this name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2305 External subprogram ″&1″ not found. Could not mark as ″low frequency″.

Where: &1 is a subprogram name.

Explanation: A ″lowfreq″ directive entry for the specified subprogram was present in the IPA Link control file, but no subprogram by this name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2306 External subprogram ″&1″ not found. Could not mark as ″an exit″.

Where: &1 is a subprogram name.

Explanation: A ″exits″ directive entry for the specified subprogram was present in the IPA Link control file, but no subprogram by this name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2307 External symbol ″&1″ not found. Could not mark as ″retain″.

Where: &1 is a symbol name.

Explanation: A ″retain″ directive entry for the specified symbol was present in the IPA Link control file, but no symbol by this name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2308 Regular expression ″&1″ error: &2.

Where: &1 is a regular expression.

Explanation: The regular expression is incorrectly specified.

User Response: Correct the regular expression ″&1″.

CCN2310 External subprogram ″&1″ not found. Could not mark as ″inline″.

Where: &1 is a subprogram name.

Explanation: An ″inline″ directive entry for the specified subprogram was present in the IPA Link control file, but no subprogram by this name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2311 EXternal subprogram ″&1″ not found. Could not mark as ″do not inline″.

Where: &1 is a subprogram name.

Explanation: A ″noinline″ directive entry for the specified subprogram was present in the IPA Link control file, but no subprogram by this name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2312 Could not inline calls from ″&1″ to ″&2″ as neither external subprogram was found.

Where: &1 and &2 are subprogram names.

Explanation: An ″inline″ directive entry for calls between the specified subprograms was present in the IPA Link control file, but no subprograms by these names are present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2313 Could not inhibit inlining calls from ″&1″ to ″&2″ as neither external subprogram was found.

Where: &1 and &2 are subprogram names.

Explanation: A ″noinline″ directive entry for calls between the specified subprograms was present in the IPA Link control file, but no subprograms by these names are present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2314 Could not inline calls from ″&1″ to ″&2″ as external subprogram ″&3″ was not found.

Where: &1, &2 and &3 are subprogram names.

Explanation: An ″inline″ directive entry for calls between the specified subprograms was present in the IPA Link control file, but no subprogram with the specified name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2315 Could not inhibit inlining calls from ″&1″ to ″&2″ as external subprogram ″&3″ was not found.

Where: &1, &2 and &3 are subprogram names.

Explanation: A ″noinline″ directive entry for calls between the specified subprograms was present in the IPA Link control file, but no subprogram with the specified name is present in the application program.

User Response: Correct the IPA Link control file directive.

CCN2316 Could not find any calls from ″&1″ to ″&2″ to inline.

Where: &1 and &2 are subprogram names.

Explanation: An ″inline″ directive entry for calls between the specified subprograms was present in the IPA Link control file, but no such calls are present in the application program.

User Response: Delete the IPA Link control file directive.

CCN2317 Could not find any calls from ″&1″ to ″&2″ to inhibit from inlining.

Where: &1 and &2 are subprogram names.

Explanation: A ″noinline″ directive entry for calls between the specified subprograms was present in the IPA Link control file, but no such calls are present in the application program.

User Response: Delete the IPA Link control file directive.

CCN2320 The minimum size of partition &1 exceeds the partition size limit.

Where: &1 is the number of the current partition.

Explanation: The program information which must be contained within the current partition is larger than the current partition size limit. This may be because the partition contains a single large subprogram.

User Response: Use the IPA Link ″partsize″ directive to specify a larger partition size limit.

CCN2340 Code generation was not performed due to previously detected errors. Object file not created.

Explanation: The completion of the IPA Link step is not possible due to errors that were previously detected. The generation of code and data from the IPA object information will not be performed, and no object file will be generated.

User Response: Eliminate the cause of the error conditions.

CCN2341 Code generation for partition &1 terminated due to previous errors.

Where: &1 is the number of the current partition.

Explanation: The generation of object code and data for the current partition has been terminated due to error conditions detected during processing. Processing continues to allow further errors to be detected, but an incomplete object file will be generated.

User Response: Eliminate the cause of the error conditions.

CCN2342 Code generation for partition &1 bypassed due to previous errors.

Where: &1 is the number of the current partition.

Explanation: The generation of object code and data for the current partition has been bypassed due to error conditions detected when processing a previous partition. Processing continues to allow further errors to be detected, but an incomplete object file will be generated.

User Response: Eliminate the cause of the error conditions.

CCN2345 An error occurred during code generation. The code generation return code was &1.

Where: &1 is the code generation return code.

Explanation: During the generation of code for the current partition, an error was detected. One or more messages may be issued when this occurs.

User Response: Refer to the responses for these messages, and perform the suggested error recovery actions.

CCN2400 File ″&1″ not found.

Where: &1 is a file name.

Explanation: The compiler could not locate the specified file.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2401 Object file ″&1″ not found.

Where: &1 is an object file name.

Explanation: The compiler could not locate the specified object file.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2402 Library file ″&1″ not found.

Where: &1 is a library file name.

Explanation: The compiler could not locate the specified library file.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2403 Archive library file ″&1″ not found.

Where: &1 is an archive library file name.

Explanation: The compiler could not locate the specified archive library file.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2404 IPA Link control file ″&1″ not found.

Where: &1 is an IPA Link control file name.

Explanation: The compiler could not locate the specified IPA Link control file.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2406 Load Module library member ″&1″ not found.

Where: &1 is a Load Module library member name.

Explanation: The compiler could not locate the specified member of the Load Module library.

User Response: Ensure the member name and Load Module library names are correct. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2407 File ″&1″ not found.

Where: &1 is a file name.

Explanation: The compiler could not locate the specified file.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2408 File ″&1″ not found.

Where: &1 is a file name.

Explanation: The compiler could not locate the specified file. Processing is terminated.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2420 File ″&1″ has invalid format.

Where: &1 is a file name.

Explanation: The specified file was located, but did not have the correct format.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Correct the file as necessary and repeat the step.

CCN2421 Library file ″&1″ has invalid format.

Where: &1 is a library file name.

Explanation: The specified file was located, but did not have the correct format to be recognized as an object library.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Correct the library as necessary and repeat the step.

CCN2422 Archive library file ″&1″ has invalid format.

Where: &1 is an archive library file name.

Explanation: The specified file was located, but did not have the correct format to be recognized as an archive library.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Rebuild the archive library as necessary and repeat the step.

CCN2423 Load Module file ″&1″ has invalid format.

Where: &1 is a Load Module file name.

Explanation: The specified file was located, but did not have the correct format to be recognized as a Load Module.

User Response: Ensure the file name is correct. Correct the Load Module library as necessary and repeat the step.

CCN2425 File ″&1″ has invalid attributes.

Where: &1 is a file name.

Explanation: The specified file was located, but did not have the correct attributes.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Correct the file as necessary and repeat the step.

CCN2426 Unable to determine attributes for file ″&1″.

Where: &1 is a file name.

Explanation: The specified file was located, but the compiler was unable to determine the file attributes.

User Response: Ensure the file name is correct. If the file is located on a LAN drive, ensure the LAN is working properly. Correct the file as necessary and repeat the step.

CCN2430 File ″&1″ is not allocated.

Where: &1 is a file name.

Explanation: The specified file is not allocated, and is unavailable for processing.

User Response: Ensure the file allocation specification is correct. Correct the file allocation as necessary and repeat the step.

CCN2431 File ″&1″ is not allocated. Autocall will not be performed.

Where: &1 is a file name.

Explanation: The specified file is not allocated, and is unavailable for autocall processing.

User Response: Ensure the file allocation specification is correct. Correct the file allocation as necessary and repeat the step.

CCN2440 Unable to open file ″&1″, for read.

Where: &1 is a file name.

Explanation: The compiler could not open the specified file. This file was being opened with the intent of reading the file contents.

User Response: Ensure the file name is correct. Ensure that the correct file is being read and has not been damaged. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2441 Unable to open file ″&1″, for write.

Where: &1 is a file name.

Explanation: The compiler could not open the specified file. This file was being opened with the intent of writing new information.

User Response: Ensure the file name is correct. Ensure that the correct file is specified. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2442 An error occurred while reading file ″&1″.

Where: &1 is a file name.

Explanation: The compiler detected an error while reading from the specified file.

User Response: Ensure that the correct file is being read and has not been damaged. If the file is located on a LAN drive, ensure the LAN is working properly.

CCN2443 An error occurred while writing to file ″&1″.

Where: &1 is a file name.

Explanation: The compiler detected an error while writing to the specified file.

User Response: Ensure that the correct file is specified. If the file is located on a LAN drive, ensure the LAN is working properly.

CCN2444 Unable to close file ″&1″, after read.

Where: &1 is a file name.

Explanation: The compiler could not close the specified file after reading the file contents.

User Response: Ensure the file name is correct. Ensure that the correct file is being read and has not been damaged. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2445 Unable to close file ″&1″, after write.

Where: &1 is a file name.

Explanation: The compiler could not close the specified file after writing new information.

User Response: Ensure that sufficient space is available to contain the file data. Ensure the file name is correct. Ensure that the correct file is specified. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2446 File ″&1″ is empty.

Where: &1 is a file name.

Explanation: The compiler opened the specified file, but it was empty when an attempt was made to read the file contents.

User Response: Ensure the file name is correct. Ensure that the correct file is being read and has not been damaged. If the file is located on a LAN drive, ensure the LAN is working properly.

CCN2447 Premature end occurred while reading file ″&1″.

Where: &1 is a file name.

Explanation: The compiler opened the specified file and began processing the file contents. The end of file was reached before all data was processed. Processing continues with the next file.

User Response: Ensure that the correct file is being read and has not been damaged. If the file is located on a LAN drive, ensure the LAN is working properly.

CCN2450 Unable to remove file ″&1″.

Where: &1 is a file name.

Explanation: The compiler could not remove the specified file.

User Response: Ensure the file name is correct. Ensure that the correct file is specified. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2451 Unable to create temporary file ″&1″.

Where: &1 is a file name.

Explanation: The compiler could not create the specified temporary file.

User Response: If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission.

CCN2460 Listing file ″&1″ is full.

Where: &1 is the listing file name.

Explanation: The compiler detected that there is insufficient free space to continue writing to the listing file. Compilation continues, without further updates to the listing file.

User Response: Ensure that the correct listing file is specified, and that there is sufficient free space. If the file is located on a LAN drive, ensure the LAN is working properly.

CCN2461 Listing file ″&1″ closed prematurely.

Where: &1 is the listing file name.

Explanation: The compiler detected an error while writing to the listing file. Compilation continues, without further updates to the listing file.

User Response: Ensure that the correct listing file is specified. If the file is located on a LAN drive, ensure the LAN is working properly.

CCN2462 Unable to write to temporary file ″&1″.

Where: &1 is the temporary file name.

Explanation: The compiler detected an error while writing to the temporary file.

User Response: Ensure there is enough disk space.

CCN2463 Unable to create a temporary file.

Explanation: The compiler could not create a temporary file.

User Response: Check the system documentation on creating temporary files.

CCN2490 COMPILER LIMIT EXCEEDED: Insufficient virtual storage.

Explanation: The compiler ran out of memory attempting to compile the file. This sometimes happens with large files or programs with large functions. Note that very large programs limit the amount of optimization that can be done.

User Response: Redefine your virtual storage to a larger size. If sufficient storage is not available, you can try various approaches, such as shut down any large processes that are running, ensure your swap path is large enough, try recompiling the program with a lower level of optimization or without interprocedural analysis.

CCN2491 COMPILER ERROR: Unimplemented feature: &1.

Explanation: An error occurred during compilation.

User Response: See the C/C++ Language Reference for a description of supported features.

CCN2492 INTERNAL COMPILER ERROR: Error &1 in Procedure &2.

Explanation: An internal compiler error occurred during compilation.

User Response: Contact your Service Representative.

CCN2493 INTERNAL COMPILER ERROR: &1.

Explanation: An internal compiler error occurred during compilation.

User Response: Contact your Service Representative.

CCN2494 SYSTEM LIMIT EXCEEDED: Too many processes are active.

Explanation: The system ran out of processes during the compilation of the file.

User Response: Try recompiling with fewer competing processes, or increase the system limit.

CCN2496 Unable to create thread.

Explanation: The system was unable to create a new thread.

User Response: Ensure there are enough system resources to support multi-threading, and that the system supports threads.

CCN3001 INTERNAL COMPILER ERROR: Procedure &1.

Explanation: An internal compiler error occurred during compilation.

User Response: Contact your VisualAge for C++ Service Representative.

CCN3002 COMPILER ERROR: Unimplemented feature: &1.

Explanation: An error occurred during compilation.

User Response: See the C/C++ Language Reference for a description of supported features.

CCN3003 Width of a bit field of type ″&1″ cannot exceed &2.

Explanation: The length of the bit field must not exceed the maximum bit size of the bit field's type.

User Response: Define the bit-field length to be less than or equal to the maximum bit size of the bit-field type.

CCN3004 #pragma must appear before use of identifier &1. #pragma ignored.

Explanation: The identifier is modified by the #pragma after the #pragma is seen.

User Response: Move the #pragma so that it appears before the identifier is used.

CCN3005 Error in message set &1, unable to retrieve message &2.

Explanation: Message cannot be retrieved from the message catalog.

User Response: Check the installation procedure to see if the message catalog has been properly installed.

CCN3006 Label &1 is undefined.

Explanation: A label must be visible in the current function scope if it is used in an expression.

User Response: Declare a label of that name in the current function scope.

CCN3007 ″&1″ is undefined.

Explanation: A C identifier must be declared before it is used in an expression.

User Response: Declare an identifier of that name in the current scope or in a higher scope.

CCN3008 The argument is not valid for the #pragma directive.

Explanation: #pragma does not recognize the argument.

User Response: Remove the argument or change its format.

CCN3009 Bit-field &1 must be of type signed int, unsigned int or int.

Explanation: The type of the bit-field is not a signed int, unsigned int nor an int.

User Response: Define the bit-field with a type signed int or unsigned int.

CCN3010 Macro &1 invoked with a null argument for parameter &2.

Explanation: No argument was specified for parameter.

User Response: Specify arguments for all macro parameters.

CCN3012 Operand of bitwise complement must be an integral type.

Explanation: The operand of the bitwise complement operator does not have an integral type. Valid integral types include: signed and unsigned char; signed and unsigned short, long, and int; and enum.

User Response: Change the type of the operand, or use a different operand.

CCN3013 Operand of unary + or - operator must be an arithmetic type.

Explanation: The operand of the unary + or - operator does not have an arithmetic type. Valid arithmetic types include: signed and unsigned char; signed and unsigned short, long, and int; enum, float, double, and long double.

User Response: Change the type of the operand, or use a different operand.

CCN3014 Operand of logical negation must be a scalar type.

Explanation: The operand of the logical negation operator (!) does not have a scalar type. Valid scalar types include: signed and unsigned char; signed and unsigned short, long, and int; enum, float, double, long double, and pointers.

User Response: Change the type of the operand, or use a different operand.

CCN3017 Operand of address operator must be an lvalue or function designator.

Explanation: The operand of the address operator (unary &) is not valid. The operand must be either a function designator or an lvalue that designates an object that is not a bit-field and is not declared with register storage class.

User Response: Change the operand.

CCN3018 Operand of indirection operator must be a pointer expression.

Explanation: The operand of the indirection operator (unary *) is not a pointer.

User Response: Change the operand to a pointer.

CCN3019 Expecting an array or a pointer to object type.

Explanation: Index operator ([]) operates only on arrays or pointer to objects.

User Response: Change the operand.

CCN3020 Expression must be an integral type.

Explanation: The expression does not evaluate to an integral type. Valid integral types include: signed, unsigned and plain char, signed and unsigned short, int, long, and enum.

User Response: Change the type of the operand.

CCN3021 Expecting struct or union.

Explanation: The left hand operand of the dot operator (.) must have a struct or union type.

User Response: Change the operand.

CCN3022 ″&1″ is not a member of ″&2″.

Explanation: The specified member does not belong to the structure or union given. One of the following has occurred:

1. The right hand operand of the dot (.) operator is not a member of the structure or union specified on the left hand side of the operator.
2. The right hand operand of the arrow (->) operator is not a member of the structure or union pointed to by the pointer on the left hand side of the operator.

User Response: Change the identifier.

CCN3023 Expecting function or pointer to function.

Explanation: The expression is followed by an argument list but does not evaluate to a function designator.

User Response: Change the expression to be a function or a pointer to a function.

CCN3024 The operand of the __alignof operator is not valid.

Explanation: The __alignof operator cannot be used with functions, void types, bit fields, incomplete types, or arrays of unknown size. The alignof operator cannot be applied to an expression that has a function type or an incomplete type, to the parenthesized name of such a type, or to an lvalue that designates a bit-field object.

User Response: Change the operand.

CCN3025 Operand must be a modifiable lvalue.

Explanation: A modifiable lvalue is an expression representing an object that can be changed.

User Response: Change the operand.

CCN3026 Number of initializers cannot be greater than the number of aggregate members.

Explanation: Too many initializers were found in the initializer list for the indicated declaration.

User Response: Check the number of initializers and change it to correspond to the number of declared members. Make sure the closing brace at the end of the initializer list is positioned correctly.

CCN3027 Function &1 cannot be initialized.

Explanation: An attempt was made to assign an initial value to a function identifier. You can not assign a value to a function identifier.

User Response: Remove the assignment operator and the initializer.

CCN3028 Storage class ″&1″ cannot be used with external data.

Explanation: The storage class is not appropriate for this declaration. Restrictions include: 1) Storage class specifier not allowed on aggregate members, casts, sizeof or offsetof declarations. 2) Declarations at file scope cannot have ’register’ or ’auto’ storage class.

User Response: Specify a different storage class.

CCN3029 #pragma ignored, identifiers are already disjoint.

Explanation: The identifiers that are specified in the pragma are already known to be disjoint so the pragma is ignored.

User Response: Nothing, or remove the pragma as it is redundant.

CCN3030 **Identifier &1 cannot be redeclared.**

Explanation: The identifier has already been declared.

User Response: Remove one of the declarations.

CCN3031 **All dimensions except the first must be specified for a multi-dimensional array.**

Explanation: Only the first dimension of an initialized array can be unspecified. All the other dimensions must be specified on the declaration.

User Response: Specify all the other dimensions in the array declaration.

CCN3032 **Elements of an array cannot be functions.**

Explanation: An array must be composed of elements that are an object type. Functions are not object types and thus cannot be elements of an array.

User Response: Use a pointer to the function, or change the type of the element.

CCN3033 **Function &1 is not valid. Function cannot return a function.**

Explanation: A function cannot have a return type of function.

User Response: Return a pointer to the function or specify a different return type.

CCN3034 **Function &1 is not valid. Function cannot return an array.**

Explanation: A function cannot return an array and the specified return type of the function is an array.

User Response: Return a pointer to the array or specify a different return type.

CCN3035 **Storage class ″&1″ cannot be used with functions.**

Explanation: A function can only have a storage class of extern or static.

User Response: Remove the storage class specifier for the function identifier, or change it to either extern or static.

CCN3036 **Range error.**

Explanation: The value is outside of the valid range.

User Response: Change value to be within the required limits.

CCN3037 **Member of struct or union cannot be a function.**

Explanation: Members of structs or unions must have object type. Functions do not have object type and cannot be members of a struct or union.

User Response: Use a pointer to the function or remove the function from the member list.

CCN3039 **Expecting a parameter after # operator.**

Explanation: The # preprocessor operator can only be applied to a macro parameter.

User Response: Place a parameter after the # token, or remove the token.

CCN3041 **The invocation of macro &1 contains fewer arguments than required by the macro definition.**

Explanation: The number of arguments supplied to the macro must match the number of parameters in the macro definition. There are not enough arguments supplied.

User Response: Complete the specification of the macro argument list.

CCN3043 **The operand of the sizeof operator is not valid.**

Explanation: Sizeof operator cannot be used with functions, void types, bit fields, incomplete types, or arrays of unknown size. The sizeof operator cannot be applied to an expression that has a function type or an incomplete type, to the parenthesized name of such a type, or to an lvalue that designates a bit-field object.

User Response: Change the operand.

CCN3044 **Expression must be a non-negative integer constant.**

Explanation: The supplied expression must evaluate to a non-negative integer constant.

User Response: Change the constant expression to yield a non-negative value.

CCN3045 **Undeclared identifier &1.**

Explanation: You must declare a C identifier before you use it in an expression.

User Response: Declare an identifier of that name in the current scope or in a higher scope.

CCN3046 Syntax error.

Explanation: See the C/C++ Language Reference for a complete description of C syntax rules.

User Response: Correct the syntax error and compile again.

CCN3047 Incorrect hexadecimal escape sequence \x. \ ignored.

Explanation: \x is used to indicate an hexadecimal escape sequence but the sequence immediately following is not a valid hexadecimal number.

User Response: Change the sequence to a valid hexadecimal number.

CCN3048 Unable to initialize source conversion from codepage &1 to codepage &2.

Explanation: An error occurred when attempting to convert source between the codepages specified.

User Response: Ensure the codepages are correct and that conversion between these codepages is supported.

CCN3049 The object &1 has a size &2 which exceeds the compiler limit &3.

Explanation: The size of the object is too large for the compiler to represent internally.

User Response: Reduce the size of the object.

CCN3050 Return type ″&1″ in redeclaration is not compatible with the previous return type ″&2″.

Explanation: The second declaration of the function declares a different return type from the first. The declaration must be identical. When you redeclare a function, the return type and parameter types must be the same in both declarations.

User Response: Change the declaration of one or both functions so that their return types are compatible.

CCN3051 Case expression must be a valid integral constant.

Explanation: The expression in the case statement must be a constant integral expression. Valid integral expressions are: char, signed and unsigned int, and enum.

User Response: Change the expression.

CCN3052 Duplicate case label for value &1. Labels must be unique.

Explanation: Two case labels in the same switch statement cannot evaluate to the same integer value.

User Response: Change one of the labels.

CCN3053 Default label cannot be placed outside a switch statement.

Explanation: A statement is labeled with default, which can only be used as a statement label within a switch statement.

User Response: Remove the default case label, or place it inside a switch statement. Check for misplaced braces on a previous switch statement.

CCN3054 Switch statement cannot contain more than one default label.

Explanation: Only one default label is allowed within a switch statement. Nested switch statements may each have one default label. This error may have been caused by a default label that is not properly placed within a nested switch statement.

User Response: Remove one of the default labels or check for misplaced braces on nested switch statements..

CCN3055 Case label cannot be placed outside a switch statement.

Explanation: Case labels are only allowed within a switch statement.

User Response: Remove the case label, or place it within a switch statement group. Check for misplaced braces on the previous switch statement.

CCN3056 Break statement cannot be placed outside a while, do, for, or switch statement.

User Response: Remove the break statement or place it inside a while, do, for or switch statement. Check for misplaced braces on a previous statement.

CCN3057 Continue cannot be placed outside a while, do, or for statement.

Explanation: Continue is only valid as, or within, a loop body.

User Response: Remove the continue statement or place it inside a while, do or for loop. Check for misplaced braces on a previous loop.

CCN3058 Label &1 has already been defined on line &2 of ″&3″.

Explanation: You already used the label to identify a section of code in the file indicated. You cannot redefine a label.

User Response: Change the name of one of the labels.

CCN3059 Comment that started on line &1 must end before the end of file.

Explanation: A comment that was not terminated has been detected. The comment started on the line indicated.

User Response: End the comment before the file ends.

CCN3062 Escape sequence &1 is out of the range 0-&2. Value is truncated.

Explanation: Character constants specified in an escape sequence exceeded the decimal value of 255, or the octal equivalent of 377, or the hexadecimal equivalent of FF.

User Response: Change the escape sequence so that the value does not exceed the maximum value.

CCN3067 A struct or union can only be assigned to a compatible type.

Explanation: The assignment is invalid between the given aggregate types.

User Response: Change the operands so that they have the same type.

CCN3068 Operation between types ″&1″ and ″&2″ is not allowed.

Explanation: The operation specified is not valid between the operands having the given types.

User Response: Either change the operator or the operands.

CCN3070 Register is the only storage class that can be used with parameters.

User Response: Remove the storage class specified in the parameter declaration or use the register storage class.

CCN3073 Empty character constant.

Explanation: An empty character constant is not valid. There must be at least one character between the single quotation marks.

User Response: Put at least one character inside the pair of single quotation marks.

CCN3076 Character constant &1 has more than one character. No more than rightmost 4 characters are used.

Explanation: A character constant can only have up to four bytes.

User Response: Change the character constant to contain four bytes or less.

CCN3077 The wchar_t value &1 is not valid.

Explanation: The value is not a valid wchar_t value. See the C/C++ Language Reference for information on wide characters.

User Response: Change character to a valid wchar_t. See the C/C++ Language Reference for information about the wchar_t type.

CCN3078 #&1 directive has no effect.

Explanation: A preprocessor directive has been specified that has no effect.

User Response: Remove the preproccessor directive.

CCN3085 Predefined macro &1 cannot be undefined.

Explanation: The macro is predefined. You cannot undefine predefined macros.

User Response: Remove the statement that undefines the macro.

CCN3095 Unexpected parameter &1.

Explanation: A parameter was declared in the parameter declaration list of the K&R function definition. The parameter did not appear in the parameter identifier list. It is also possible that the K&R function definition had more parameters than the function prototype.

User Response: Change the number of parameters.

CCN3098 Missing argument(s).

Explanation: The function call contains fewer arguments than specified in the parameter list of the function prototype.

User Response: Make sure the function call has the same number of arguments as the function prototype has parameters.

CCN3099 Unexpected argument.

Explanation: The function call contains more arguments than specified in the parameter list of the function prototype.

User Response: Change the number of arguments in the function call or change the function prototype.

CCN3103 Tag &1 requires a complete definition before it is used.

Explanation: Only pointer declarations can include incomplete types. A struct or union tag is undefined if the list describing the name and type of its members has not been specified.

User Response: Define the tag before it is used in the declaration of an identifier or complete the declaration.

CCN3104 The value of an enumeration constant must be an integral constant expression.

Explanation: If an enum constant is initialized in the definition of an enum tag, the initial value of the constant must be an integral expression that has a value representable as an int.

User Response: Remove the initial value, or ensure that the initial value is an integral constant expression with a value representable as an int.

CCN3108 Bit fields with zero width must be unnamed bit fields.

Explanation: A named bit field must have a positive length; a zero length bit field is used for alignment only and must not be named.

User Response: Redefine the bit field with a length greater than zero or remove the name of the bit-field.

CCN3112 Duplicate type qualifier ″&1″ ignored.

Explanation: The indicated qualifier appears more than once in the type declaration.

User Response: Remove one of the duplicate qualifiers.

CCN3115 Duplicate type specifier ″&1″ ignored.

Explanation: A duplicate type specifier appears in the type declaration.

User Response: Remove one of the duplicate type specifiers.

CCN3117 Operand must be a scalar type.

Explanation: Valid scalar types include: signed and unsigned char; signed and unsigned short, long, and int; enum, float, double, long double, and pointers.

User Response: Change the type of the operand, or use a different operator.

CCN3119 Duplicate storage class specifier &1 ignored.

Explanation: A duplicate storage class specifier appears in the declaration.

User Response: Remove one of the duplicate storage class specifiers.

CCN3120 Function cannot return a &1 qualified type.

Explanation: The const or volatile qualifier cannot be used to qualify a function’s return type.

User Response: Remove the qualifier or return a pointer to the qualified type.

CCN3122 Expecting pointer to struct or union.

Explanation: The left hand operand of the arrow operator (->) must have type pointer to structure or pointer to union.

User Response: Change the operand.

CCN3127 The second and third operands of the conditional operator must have compatible struct or union types.

Explanation: If one operand in the conditional expression has type struct or union, the other operand must also have type struct or union.

User Response: Make the operands compatible.

CCN3131 Explicit dimension specification or initializer required for an auto or static array.

Explanation: For arrays of automatic or static storage class, all dimensions of the array must be specified in the declaration. If the declaration provides an initialization, the first dimensions may be unspecified because the initialization will determine the size needed.

User Response: Specify all of the dimensions in the array declaration.

CCN3134 Array bound is too large.

Explanation: The size of the array is too large for the compiler to represent internally.

User Response: Reduce the size of the array.

CCN3137 Declaration must declare at least one declarator, tag, or the members of an enumeration.

Explanation: The declaration specifier was the only component of the declaration. eg. int ;

User Response: Specify at least one declarator, tag, or member of an enumeration.

CCN3152 A register array may only be used as the operand to sizeof.

Explanation: The only operator that can be applied to an array declared with storage class specifier register is sizeof.

User Response: Remove the operation or remove the register storage class specifier.

CCN3155 Option &1 requires suboption(s).

Explanation: The option is not completely specified; a suboption is required.

User Response: Add a suboption.

CCN3159 Bit-field type specified for &1 is not valid. Type &2 assumed.

Explanation: The type of a bit-field must be a (possibly qualified) version of int, signed int or unsigned int.

User Response: Define the bit-field with a type signed int or unsigned int.

CCN3160 Object &1 cannot be declared as type void.

Explanation: The type void can only be used as the return type or parameter list of a function, or with a pointer. No other object can be of type void.

User Response: Ensure that the declaration uses type void correctly.

CCN3162 No definition was found for function &1. Storage class changed to extern.

Explanation: A static function was declared and referenced in this file. The definition of the function was not found before the end of the file. When a function is declared to be static, the function definition must appear in the same file.

User Response: Change the storage class to extern or provide a function definition in this file.

CCN3164 Expression must be a scalar type.

Explanation: Valid scalar types include: signed and unsigned char; signed and unsigned short, long, and int; enum, float, double, long double, and pointers.

User Response: Change the expression.

CCN3166 Definition of function &1 requires parentheses.

Explanation: The syntax of the declaration is not correct. The compiler assumes it is the declaration of a function in which the parentheses surrounding the parameters are missing.

User Response: Check the syntax of the declaration. Ensure the object name and type are properly specified. Check for incorrect spelling or missing parentheses.

CCN3167 String literal is longer than target array. Literal is truncated on the right.

Explanation: An attempt was made to initialize an array with a string that is too long. The largest possible prefix of the string has been placed in the array.

User Response: Increase the size of the array. Make sure you include space for the terminating null character.

CCN3168 Initializer must be enclosed in braces.

Explanation: The initializer list for a declarator must be enclosed in braces.

User Response: Check for misplaced or missing braces.

CCN3169 Too many suboptions specified for option FLAG. Specify only two suboptions.

Explanation: The FLAG option takes two suboptions separated by ':'. The suboptions indicate the level of errors to be reported in the source listing and in stderr.

User Response: Only specify two suboptions to the FLAG option.

CCN3170 Parameter &1 has already been defined on line &2 of ″&3″.

Explanation: A parameter can only be defined once but more than one definition for the parameter has been specified. Parameters names must be unique.

User Response: Remove one of the parameter declarations or change the name of the identifier.

CCN3172 Parameter type list for function &1 contains parameters without identifiers.

Explanation: In a C function definition, all parameters must be named in the parameter list. The only exceptions are parameters of type void.

User Response: Name the parameter or remove it.

CCN3173 Option &1 is not recognized.

Explanation: An invalid option was specified.

User Response: Correct the spelling of the option.

CCN3174 Option &1 must be specified on the command line.

Explanation: The option can only be specified on the command line and is not valid as part of an options pragma.

User Response: Specify option on command line.

CCN3175 Option &1 must be specified on the command line or before the first C statement in the program.

Explanation: The option is specified in a pragma options after the first C token in the compilation unit. It must be specified before the first token.

User Response: Specify the option on the command line or move the pragma options before the first token.

CCN3176 Option &1 cannot take more than one suboption.

Explanation: More than one suboption was specified for an option that can only accept one suboption.

User Response: Remove the extra suboptions.

CCN3177 Type combination is not valid.

CCN3178 Unexpected argument for built-in function &1.

Explanation: The function call contains more arguments than specified in the parameter list of the built-in function.

User Response: Change the number of arguments in the function call.

CCN3180 Redeclaration of built-in function &1 ignored.

Explanation: Built-in functions are declared by the compiler and cannot be redeclared.

User Response: Remove the declaration.

CCN3181 Definition of built-in function &1 ignored.

Explanation: Built-in functions are defined by the compiler and cannot be redefined.

User Response: Remove the function definition.

CCN3182 Arguments missing for built-in function &1.

Explanation: The function call contains fewer arguments than specified in the parameter list of the built-in function.

User Response: Change the number of arguments in the function call.

CCN3183 Builtin function &1 cannot change a read-only string literal.

Explanation: Read-only strings cannot be modified.

User Response: Modify a copy of the string or change the string’s read-only status.

CCN3184 Too few suboptions specified for option FLAG. Specify two suboptions.

Explanation: The FLAG option takes two suboptions separated by ’:’. The suboptions indicate the level of errors to be reported in the source listing and in stderr.

User Response: Specify two suboptions to the FLAG option.

CCN3185 #line number &1 must be greater than zero.

Explanation: The #line directive tells the compiler to treat the following source lines as starting from the specified line. This number must be a non-negative offset from the beginning of the file.

User Response: Change line number to a non-negative integer.

CCN3186 String literal must be ended before the end of line.

Explanation: String literals must end before the end of the line. To create a string literal longer than one line, use the line continuation sequence (a backslash (\) at the end of the line), or concatenate adjacent string literal.

User Response: End the string with a quotation mark before the end of the line or use the continuation sequence.

CCN3188 Reserved name &1 cannot be defined as a macro name.

Explanation: The name is reserved for the compiler's use.

User Response: Choose another name.

CCN3189 Floating point constant &1 is not valid.

Explanation: See the C/C++ Language Reference for a description of a floating-point constant.

User Response: Ensure that the floating-point constant does not contain any characters that are not valid.

CCN3190 Automatic constant &1 does not have a value. Zero is being assumed.

Explanation: Const qualified variable declarations should contain an initializer. Otherwise you cannot assign the variable a value.

User Response: Initialize the const variable when you declare it.

CCN3191 The character &1 is not a valid C source character.

Explanation: Refer to the C/C++ Language Reference for information on valid characters.

User Response: Change the character.

CCN3192 Cannot take address of built-in function &1.

Explanation: You cannot take the address of a built-in function or declare a pointer to a built-in function.

User Response: Remove the operation that takes the address of the built-in function.

CCN3193 The size of this type is zero.

Explanation: You cannot take the address of an array of size zero.

User Response: Remove the operation that takes the address of the zero-sized array.

CCN3194 Incomplete type is not allowed.

Explanation: Except for pointers, you cannot declare an object of incomplete type.

User Response: Complete the type declaration.

CCN3195 Integral constant expression with a value greater than zero is required.

Explanation: The size of an array must be an expression that evaluates to a compile-time integer constant that is larger than zero.

User Response: Change the expression.

CCN3196 Initialization between types ″&1″ and ″&2″ is not allowed.

Explanation: An attempt was made to initialize a variable with an incompatible type.

User Response: Ensure types are compatible.

CCN3197 Expecting header file name in #include directive.

Explanation: There was no header filename after the #include directive.

User Response: Specify the header file name. Enclose system header names in angle brackets and user header names in double quotes.

CCN3198 #if, #else, #elif, #ifdef, #ifndef block must be ended with #endif.

Explanation: Every #if, #ifdef, and #ifndef must have a corresponding #endif.

User Response: End the conditional preprocessor statements with a #endif.

CCN3199 #&1 directive requires a macro name.

Explanation: There must be a macro name after every #define, #undef, #ifdef or #ifndef.

User Response: Ensure that a macro name follows the #define, #undef, #ifdef, or #ifndef preprocessor directive.

CCN3200 #elif can only appear within a #if, #elif, #ifdef, or #ifndef block.

Explanation: #elif is only valid within a conditional preprocessor block.

User Response: Remove the #elif statement, or place it within a conditional preprocessor block.

CCN3201 #else can only appear within a #if, #elif, #ifdef or #ifndef block.

Explanation: #else is only valid within a conditional preprocessor block.

User Response: Remove the #else statement, or place it within a conditional preprocessor block.

CCN3202 #endif can only appear at the end of a #if, #elif, #ifdef or #ifndef block.

Explanation: Every #endif must have a corresponding #if, #ifdef, or #ifndef.

User Response: Remove the #endif statement, or place it after a conditional preprocessor block.

CCN3204 Unexpected end of file.

Explanation: The end of the source file has been encountered prematurely.

User Response: Check for misplaced braces.

CCN3205 &1

Explanation: The #error directive was encountered. Compilation terminated.

User Response: Recompile with correct macro definitions.

CCN3206 Suffix of integer constant &1 is not valid.

Explanation: Valid integer suffixes are u or U for unsigned, or l or L for long. Unsuffixed constants are given the smallest data type that can hold the value. Refer to the C/C++ Language Reference.

User Response: Change or remove the suffix.

CCN3207 Integer constant &1 out of range.

Explanation: The specified constant is too large to be represented by an unsigned long int.

User Response: The constant integer must have a value less than UINT_MAX defined in <limits.h>.

CCN3208 Compilation ended due to an I/O error.

Explanation: A file read or write error occurred.

User Response: Ensure that you have read access to all source files, and read and write access to the TMP directory. You also need write access to the object output directory.

CCN3209 Character constants must end before the end of a line.

Explanation: Character literals must be terminated before the end of the line.

User Response: End the character literal before the end of the line. Check for misplaced quotation marks.

CCN3210 The ## operator requires two operands.

Explanation: The ## operator must be preceded and followed by valid tokens in the macro replacement list. Refer to the C/C++ Language Reference for information on the ## operator.

User Response: Provide both operands for the ## operator.

CCN3211 Parameter list must be empty, or consist of one or more identifiers separated by commas.

Explanation: The macro parameter list must be empty, contain a single identifier, or contain a list of identifiers separated by commas.

User Response: Correct the parameter list.

CCN3212 Duplicate parameter &2 in definition of macro &1.

Explanation: The identifiers in the macro parameter list must be unique.

User Response: Change the identifier name in the parameter list.

CCN3213 Macro name &1 cannot be redefined.

Explanation: You can define a macro multiple times only if the definitions are identical except for white space separating the tokens.

User Response: Change the macro definition to be identical to the preceding one, or remove it.

CCN3215 Too many arguments specified for macro &1.

Explanation: The number of arguments specified in the macro invocation is different from the number of parameters specified in the macro definition.

User Response: Make the number of arguments consistent with the macro definition.

CCN3218 Unknown preprocessing directive #&1.

Explanation: An unrecognized preprocessing directive has been encountered.

User Response: Check the spelling and syntax or remove the directive.

CCN3219 #line value &1 too large.

Explanation: The value for a #line directive must not exceed 32767.

User Response: Ensure that the #line value does not exceed 32767.

CCN3220 #line value &1 must contain only decimal digits.

Explanation: A non-numeric character was encountered in the #line value.

User Response: Check the syntax of the value given.

CCN3221 Initializer must be a valid constant expression.

Explanation: The initializers for objects of static storage duration, for elements of an array, or for members of a structure or union must be valid constant expressions.

User Response: Remove the initialization or change the indicated initializer to a valid constant expression.

CCN3224 Incorrect #pragma ignored.

Explanation: An unrecognized #pragma directive was encountered. See the C/C++ Language Reference for the list of valid #pragma directives.

User Response: Change or remove the #pragma directive.

CCN3225 Error reading file &1. (&2)

User Response: Ensure that the file exists and that the compiler can access it.

CCN3226 The ″:″ operator is not allowed between ″&1″ and ″&2″.

Explanation: The operands must be of compatible type.

User Response: Change the type of the operands.

CCN3229 File is empty.

Explanation: The source file contains no code.

User Response: Check that the file name and path are correct. Add source code to the file.

CCN3231 Error occurred while opening preprocessor output file.

Explanation: The preprocessor was unsuccessful in attempting to open the output file.

User Response: Ensure you have write access to the file.

CCN3232 Divisor for modulus or division operator cannot be zero.

Explanation: The value of the divisor expression cannot be zero.

User Response: Change the expression used as the divisor.

CCN3234 Expecting a new-line character on #&1 directive.

Explanation: A character sequence was encountered when the preprocessor required a new-line character.

User Response: Add a new-line character.

CCN3235 Incorrect escape sequence &1. \ ignored.

Explanation: An escape sequence that is not valid has been encountered in a string literal or a character literal. It is replaced by the character following the backslash (\).

User Response: Change or remove the escape sequence.

CCN3236 Macro name &1 has been redefined.

Explanation: You can define a macro multiple times in extended mode. In ANSI mode macro redefinitions are ignored.

User Response: Change the language level to extended (with the /Se compiler option or #pragma langlvl directive), or remove the macro redefinitions.

CCN3238 Function argument cannot be type void.

Explanation: The void type cannot appear in the argument list of a function call. The void type can appear in a parameter list only if it is a non-variable argument function. It is the only parameter in the list, and it is unnamed.

User Response: Correct the argument or remove the argument.

CCN3242 An object with external linkage declared at block scope cannot be initialized.

Explanation: You cannot declare a variable at block scope with the storage class extern and give it an explicit initializer.

User Response: Initialize the external object in the external declaration.

CCN3243 Value of enumeration constant must be in range of signed integer.

Explanation: If an enum constant is initialized in the definition of an enum tag, the initial value must be an integral expression that has a value representable as an int.

User Response: Remove the initial value, or ensure that it is an integral constant expression that has a value representable as an int.

CCN3244 External variable &1 cannot be redefined.

Explanation: An attempt was made to redefine an external variable.

User Response: Remove the redefinition.

CCN3245 Incompatible sign adjective ″&1″.

Explanation: Adjectives ″signed″ and unsigned can only modify integer type specifiers.

User Response: Either remove the sign adjective or use a different type specifier.

CCN3246 Incompatible length adjective ″&1″.

Explanation: Length adjectives short and long can only be applied to particular scalar types. See the C/C++ Language Reference for valid types.

User Response: Either remove the length adjective or use a different type specifier.

CCN3247 Incompatible type specifier ″&1″.

Explanation: The type specifier is not compatible with the type adjectives used. See the C/C++ Language Reference for valid combinations of type specifiers and adjectives.

User Response: Either remove the adjective or use a different type specifier.

CCN3248 More than one storage class specifier &1.

Explanation: A C declaration must only have one storage class specifier.

User Response: Ensure only one storage class is specified.

CCN3249 Identifier contains a $ character.

Explanation: You cannot use the $ character in an identifier. An identifier can contain alphanumeric characters and underscores. An identifier must start with either an underscore or alphabetic character.

User Response: Remove the $ character.

CCN3250 Floating point constant &1 out of range.

Explanation: The compiler detected a floating-point overflow either in scanning a floating-point constant, or in performing constant arithmetic folding.

User Response: Change the floating-point constant so that it does not exceed the maximum value.

CCN3251 Static function &1 is undefined.

Explanation: A static function was declared and referenced in this file. The definition of the function was not found before the end of the file. When a function is declared to be static, the function definition must appear in the same file.

User Response: Define the function in the file or remove the static storage class.

CCN3255 #pragma &1 is out of sequence.

Explanation: The #pragma directive was out of sequence. See the C language Reference Manual for the restrictions on placement.

User Response: Change or remove the #pragma directive.

CCN3258 Hexadecimal integer constant &1 is not valid.

Explanation: An invalid hexadecimal integer constant was specified. See the C/C++ Language Reference for details on specifying hexadecimal characters.

User Response: Change the value to a valid hexadecimal integer constant.

CCN3260 Octal integer constant &1 is not valid.

Explanation: An invalid octal integer constant was specified. See the C/C++ Language Reference for details on specifying octal characters.

User Response: Change the value to a valid octal integer constant.

CCN3261 Suboption &1 is not valid for option &2.

Explanation: An invalid suboption was specified for some option.

User Response: Change the suboption.

CCN3262 #pragma &1 must occur before first C statement in program. #pragma ignored.

Explanation: This pragma must be specified before the first C token in the input (including header files).

User Response: Place the #pragma directive in the file before any C code, or remove it.

CCN3263 #pragma strings directive can be specified only once per source file. #pragma ignored.

Explanation: This #pragma specifies whether string literals are placed in read-only memory. It must appear only once and before any C code.

User Response: Change the location of the directive and ensure that it appears only once in the translation unit.

CCN3264 #pragma comment(copyright) directive can be specified only once per source file.

Explanation: There can only be one #pragma comment(copyright) per source file.

User Response: Ensure that it occurs only once in the translation unit.

CCN3266 Parameter(s) for #pragma are out of range.

Explanation: The #pragma parameters were invalid. See the C/C++ Language Reference for details on valid #pragma parameters.

User Response: Change the parameter.

CCN3267 Unrecognized #pragma ignored.

Explanation: An invalid pragma was encountered and ignored.

User Response: Ensure that the #pragma name is spelled correctly. A #pragma with equivalent function, but a different name may exist. See the C/C++ Language Reference for a list of #pragma directives.

CCN3268 Macro &1 invoked with an incomplete argument for parameter &2.

Explanation: The parameter for the macro invocation must have a complete argument.

User Response: Complete the specification of the macro argument list. Check for missing commas.

CCN3269 A char array pointer cannot be assigned to a nonchar pointer.

CCN3270 A wide char array pointer cannot be assigned to a nonwide char pointer.

CCN3271 The indirection operator cannot be applied to a void pointer.

Explanation: The indirection operator requires a pointer to a complete type. A void pointer is an incomplete type that can never be completed.

User Response: Cast the pointer to a type other than void before this operation.

CCN3272 Identifier not allowed in cast or sizeof declarations.

Explanation: Only abstract declarators can appear in cast or sizeof expressions.

User Response: Remove the identifier from the cast or sizeof expression and replace it with an abstract declarator.

CCN3273 Missing type in declaration of &1.

Explanation: A declaration was made without a type specifier.

User Response: Insert a type specifier into the declaration.

CCN3274 Missing declarator in structure member declaration.

Explanation: A struct or union member declaration must specify a name. A type cannot be followed by a semicolon.

User Response: Declare the member with a name.

CCN3275 Unexpected text &1 encountered.

Explanation: A syntax error has occurred. This message lists the tokens that were discarded by the parser when it tried to recover from the syntax error.

User Response: Correct the syntax error and compile again.

CCN3276 Syntax error: possible missing &1?

Explanation: A syntax error has occurred. This message lists the token that the parser expected and did not find.

User Response: Correct the syntax error and compile again.

CCN3277 Syntax error: possible missing &1 or &2?

Explanation: A syntax error has occurred. This message lists the tokens that the parser expected and did not find.

User Response: Correct the syntax error and compile again.

CCN3278 The structure definition must specify a member list.

Explanation: The declaration of a struct or a union that includes an empty member list enclosed between braces is not a valid struct or union definition.

User Response: Specify the members of the struct or union in the definition or remove the empty braces to make it a simple struct or union tag declaration.

CCN3279 A function declarator cannot have a parameter identifier list if it is not a function definition.

Explanation: A function declarator that is not also a function definition may not have a K&R style parameter identifier list. An example is the ″x,y″ in ″int (*fred(a,b)) (x,y) {}″.

User Response: Remove the parameter identifier list.

CCN3280 Function argument assignment between types ″&1″ and ″&2″ is not allowed.

Explanation: The type of the argument in the function call should match the corresponding parameter type in the function declaration.

User Response: Cast the argument to a different type, change the type or change the function prototype.

CCN3281 Prefix and postfix increment and decrement operators cannot be applied to ″&1″.

Explanation: Increment and decrement operators cannot operate on pointers to function or pointers to void.

User Response: Change the pointer to point to an object type.

CCN3282 The type of the parameters must be specified in a prototype.

Explanation: A prototype specifies the number and the type of the parameters that a function requires. A prototype that does not specify the type of the parameters is not correct, for example,

```
fred(a,b);
```

User Response: Specify the type of the parameters in the function prototype.

CCN3283 Functions cannot be declared &1 at block scope, &2 is ignored.

Explanation: Functions declared at block scope can only have extern as an explicit storage class specifier and cannot be inline.

User Response: Place the declaration of the function at file scope, or remove the storage class specifier or the inline specifier.

CCN3285 The indirection operator cannot be applied to a pointer to an incomplete struct or union.

Explanation: A structure or union type is completed when the definition of its tag is specified. A struct or union tag is defined when the list describing the name and type of its members is specified.

User Response: Complete the struct or union definition.

CCN3286 A struct or union with no named members cannot be explicitly initialized.

Explanation: Only aggregates containing named members can be explicitly initialized.

User Response: Name the members of the struct or union.

CCN3287 The parameter list on the definition of macro &1 is not complete.

Explanation: There is a problem with the parameter list in the definition of the macro.

User Response: Complete the parameter list. Look for misplaced or extra commas.

CCN3288 Expecting file name or new-line character on #line directive.

Explanation: The #line directive requires a line number argument as its first parameter and a file name as an optional second parameter. No other arguments are allowed. A new-line character must be present after the argument list.

User Response: Change the directive syntax.

CCN3289 Macro &1 redefined with identical definition.

Explanation: Identical macro redefinitions are allowed but not necessary. The amount of white space separating the tokens have no bearing on whether macros are considered identical.

CCN3290 Unknown macro name &1 on #undef directive.

Explanation: An attempt is being made to undefine a macro that has not been previously defined.

User Response: Check the spelling of the macro name or remove the #undef directive.

CCN3291 Expecting decimal constant on #line directive.

Explanation: The value for a #line directive must be a decimal constant.

User Response: Specify a line number on the #line directive.

CCN3292 Multibyte character literal not allowed on #&1 directive.

Explanation: The directive does not allow a multibyte character literal.

User Response: Remove the multibyte character literal.

CCN3293 Identifier &1 assigned default value of zero on &2 directive.

Explanation: The indicated identifier in a #if or #elif expression was assigned the default value of zero. The identifier may have been intended to be expanded as a macro.

User Response: Add a #define for the macro before using it in a preprocessor conditional.

CCN3294 Syntax error in expression on #&1 directive.

Explanation: The expression for a preprocessor directive contains a syntax error.

User Response: Replace the expression that controls the directive by a constant integral expression.

CCN3295 File ended with a continuation sequence.

Explanation: The file ended unexpectedly with a backslash character followed by a new-line character.

User Response: Remove the continuation character from the last line of the file, or add code after the continuation character.

CCN3296 #include file &1 not found.

Explanation: The file specified on the #include directive could not be found. See the C/C++ Language Reference for file search order.

User Response: Ensure the #include file name and the search path are correct.

CCN3297 Unable to open input file &1. (&2)

Explanation: The compiler was unable to open the input file.

User Response: Ensure file exists and is accessible by compiler.

CCN3298 Unable to read input file &1. (&2)

Explanation: The compiler was unable to read the input file.

User Response: Ensure file exists and is accessible by compiler.

CCN3299 Maximum #include nesting depth of &1 has been exceeded.

Explanation: The included files have been nested too deeply.

User Response: Reduce the number of nested include files.

CCN3300 Insufficient storage available.

Explanation: The compiler ran out of memory trying to compile the file. This sometimes happens with large files or programs with large functions. Note that very large programs limit the amount of optimization that can be done.

User Response: Increase your region size on MVS, or your virtual storage on VM. You can also divide the file into several small sections or shorten the function.

CCN3301 Redeclaration cannot specify fewer parameters than previous declaration.

Explanation: The function definition has fewer parameters than the prototype.

User Response: Modify one of the function declarations so that the number and types of the parameters match.

CCN3302 The declarations of the function &1 must be consistent in their use of the ellipsis.

Explanation: The prototyped redeclaration of the function is not correct. Fewer parameters appear before the ellipsis in this function redeclaration than the previous declaration.

User Response: Ensure that the redeclaration is consistent with the previous declaration.

CCN3303 The type of the parameter &1 cannot conflict with the previous declaration of function &2.

Explanation: Nonprototype function declarations, popularly known as K&R prototypes, specify only the function return type. The function parentheses are empty; no information about the parameters is given.

Nonprototype function definitions specify a list of parameter names appearing between the function parentheses followed by a list of declarations (located between the parentheses and the opening left brace of the function) that indicates the type of the parameters. A nonprototype function definition is also known as a K&R function definition.

A prototype function declaration or definition specifies the type and the number of the parameters in the parameter declaration list that appears inside the function parenthesis. A prototype function declaration is better known as an ANSI prototype, and a prototype function definition is better known as an ANSI function definition.

When the nonprototype function declarations/definitions are mixed with prototype declarations, the type of each prototype parameter must be compatible with the type that results from the application of the default argument promotions.

Most types are already compatible with their default argument promotions. The only ones that aren't are `char`, `short`, and `float`. Their promoted versions are, respectively, `int`, `int`, and `double`.

This message can occur in several situations. The most common is when mixing ANSI prototypes with K&R function definitions. If a function is defined using a K&R-style header, then its prototype, if present, must specify widened versions of the parameter types. Here is an example.

```
 int fn(short); int fn(x)
short x; {}
```

This is not valid because the function has a K&R-style definition and the prototype does not specify the widened version of the parameter. To be correct, the prototype should be

```
int fn(int);
```

because `int` is the widened version of `short`.

Another possible solution is to change the function definition to use ANSI syntax. This particular example would be changed to

```
 int fn(short); int fn(short x) {}
```

This second solution is preferable, but either solution is equally valid.

User Response: Give a promoted type to the parameter in the prototype function declaration.

CCN3304 No function prototype given for '&1'.

Explanation: A prototype declaration of the function specifying the number and type of the parameters was not found before the function was used. Errors may occur if the function call does not respect the function definition.

User Response: Add an appropriate function prototype before calling the function.

CCN3306 Subscript operator requires an array operand in the offsetof macro.

Explanation: A subscript was specified in the offsetof macro but the operand is not an array.

User Response: Either change the operand to be an array type or remove the subscript operator.

CCN3307 Array index must be a constant expression in the offsetof macro.

Explanation: The offsetof macro is evaluated at compile time. Thus all arguments must be constant expressions.

User Response: Change the expression.

CCN3308 Operand of the offsetof macro must be a struct or a union.

Explanation: The first operand of the offsetof macro must be a structure or union type.

User Response: Change the operand.

CCN3309 The offsetof macro cannot be used with an incomplete struct or union.

Explanation: An incomplete struct or union is not a valid argument to the offsetof macro. A structure or union type is completed when the definition of its tag is specified.

User Response: Ensure the struct or union is a complete type.

CCN3310 The type ″&1 &2″ was introduced in a parameter list, and will go out of scope at the end of the function declaration or definition.

Explanation: The tag will be added to parameter scope in ANSI mode. Thus it will go out of scope at the end of the declaration or function definition. In extended mode, the tag is added to the closest enclosing block scope.

User Response: If the tag is needed for declarations outside its scope, move the tag declaration outside of parameter scope.

CCN3311 Wide character constant &1 has more than one character. Last character is used.

Explanation: All but the last character in the constant will be discarded.

User Response: Remove all but one character or change the character constant into a string literal.

CCN3312 Compiler internal name &1 has been defined as a macro.

Explanation: Do not redefine internal compiler names.

User Response: Remove the macro definition or change the name of the macro being defined.

CCN3313 Compiler internal name &1 has been undefined as a macro.

Explanation: Do not redefine internal compiler names.

User Response: Remove the macro undefinition.

CCN3314 The tag of this expression's type has gone out of scope.

Explanation: The tag used in the type declaration of the object has gone out of scope, however the object is still referenced in the expression.

User Response: Either remove the reference to the object or move the tag's definition to a scope that encloses both the referenced object and the object's declaration.

CCN3320 Operation is not allowed because the size of &1 is unknown.

Explanation: The operand must be a complete type for the compiler to determine its size.

User Response: Provide a complete type definition.

CCN3321 You can specify an initializer only for the first named member of a union.

Explanation: There can only be an initializer for the first named member of a union.

User Response: Remove all union initializers other than the one attached to the first named member.

CCN3322 Illegal multibyte character &1.

Explanation: The multibyte character specified is not valid.

User Response: Correct the multibyte character.

CCN3323 ″double″ should be used instead of ″long float″.

Explanation: The type long float is not valid; it is treated as a double.

User Response: Remove the long type specifier or use double instead of float.

CCN3324 ″&1″ cannot be converted to ″&2″.

Explanation: The cast between the two types is not allowed.

User Response: Remove the cast.

CCN3327 An error occurred while opening the listing file, &1.

Explanation: The compiler was unable to open the listing file.

User Response: Ensure the file exists and that the compiler can access it.

CCN3328 ″″&1″ is not a valid hex digit.″

Explanation: Valid hex digits are the letters A,B,C,D,E,F,0,1,2,3,4,5,6,7,8,9.

User Response: Change the digit.

CCN3329 Byte string must have an even length.

Explanation: The byte string for a #pragma mcfunc must be of even length.

User Response: Ensure that the machine code string is of even length.

CCN3332 Option &1 is ignored because option &2 is not specified.

Explanation: The option &1 is only valid when used in conjunction with &2.

User Response: Compile with &2.

CCN3334 Identifier &1 has already been defined on line &2 of ″&3″.

Explanation: There is more than one definition of an identifier.

User Response: Remove one of the definitions or change the name of the identifier.

CCN3335 Parameter identifier list contains multiple occurrences of &1.

Explanation: Identifier names in a parameter list must be unique.

User Response: Change the name of the identifier or remove the parameter.

CCN3339 A character string literal cannot be concatenated with a wide string literal.

Explanation: A string that has a prefix L cannot be concatenated with a string that is not prefixed. Concatenation requires that both strings be of the same type.

User Response: Check the syntax of the value given.

CCN3341 #include header must be ended before the end of the line.

Explanation: A #include directive was specified across two or more lines.

User Response: Ensure that the #include directive and its arguments are contained on a single line.

CCN3342 ″″/*″ detected in comment.″

Explanation: You can ignore this message if you intended ″/*″ to be part of the comment. If you intended it to start a new comment, move it out of the enclosing comment.

User Response: Remove ″/*″ or ensure that ″/*″ was intended in the comment.

CCN3343 Redeclaration of &1 differs from previous declaration on line &2 of ″&3″.

Explanation: The redeclaration is not compatible with the previous declaration.

User Response: Either remove one declaration or make the types compatible.

CCN3344 Member &1 has already been defined on line &2 of ″&3″.

Explanation: Member names must be unique within the same aggregate.

User Response: Change the name.

CCN3345 The data in precompiled header file &1 does not have the correct format.

Explanation: The precompiled header file may have become corrupt and is ignored.

User Response: Regenerate the precompiled header files.

CCN3346 Unable to open precompiled header file &1 for input. The original header will be used.

Explanation: The compiler was unable to open the precompiled header file for reading and will use the original header.

User Response: Regenerate the precompiled header files.

CCN3347 Precompiled header file &1 was created by a more recent release of the compiler. The original header will be used.

Explanation: The compiler cannot understand the format of the precompiled header, since it was generated using a more recent version of the compiler. The original text version of the header will be used.

User Response: Regenerate the precompiled header files.

CCN3348 Unable to write to precompiled header file &1.

Explanation: The compiler was unable to write to the precompiled header files.

User Response: Ensure that the compiler has write access to the precompiled header files.

CCN3349 Value of enumeration constant must be in range of unsigned integer.

Explanation: If an enum constant is initialized in the definition of an enum tag, the value that it is initialized to must be an integral expression that has a value representable as an int.

User Response: Remove the initial value, or ensure that it is an integral constant expression that has a value representable as an int.

CCN3350 Error writing to intermediate files. &1.

Explanation: An error occurred during compilation. Ensure the compiler has write access to the work files and that there is enough space free.

User Response: Recompile compilation unit.

CCN3351 Error opening intermediate files.

Explanation: An error occurred during compilation. Ensure the compiler has write access to the work files and that there is enough space free.

User Response: Recompile compilation unit.

CCN3352 Incompatible specifications for options arch and tune.

Explanation: The values specified for tune option cannot be smaller than that of arch.

User Response: Change option values.

CCN3356 Compilation unit is empty.

Explanation: There is no code in the compilation unit.

User Response: Ensure the correct source file is specified. Recompile.

CCN3357 Unable to generate prototype for ″&1″ because one or more enum, struct, or union specifiers did not have a tag.

Explanation: A prototype could not be generated for the function because the enum, struct or union declaration did not have a tag.

User Response: Specify a tag.

CCN3358 ″&1″ is defined on line &2 of &3.

Explanation: This message indicates where a previous definition is located.

User Response: Remove one of the definitions or change the name of the identifier.

CCN3359 Automatic variable &1 contains a const member and is not initialized. It will be initialized to zero.

Explanation: An automatic variable that has a const member is not initialized. The compiler is using zero as the initializer.

User Response: Initialize the const member.

CCN3360 Same #pragma &1 has already been specified for object ″&2″; this specification is ignored.

Explanation: The repetition of the #pragma is redundant and is ignored.

User Response: Remove the duplicate #pragma.

CCN3361 A different #pragma &1 has already been specified for object ″&2″, this specification is ignored.

Explanation: A previous #pragma for the object is taking precedence over this #pragma.

User Response: Remove one of the #pragma directives.

CCN3362 Identifier ″&1″ was referenced in #pragma &2, but was never actually declared.

Explanation: A #pragma refers to an identifier that has not been declared.

User Response: Declare identifier or remove #pragma.

CCN3363 Packing boundary must be specified as one of 1, 2, 4, 8 or 16.

Explanation: Objects must be packed on 1, 2, 4, 8 or 16 byte boundaries.

User Response: Change the packing specifier.

CCN3364 main must have C calling convention.

Explanation: An inappropriate linkage has been specified for the main function. This function is the starting point of the program so only C linkage is allowed.

User Response: Change the calling convention of main.

CCN3366 Declaration cannot specify multiple calling convention specifiers.

Explanation: A declaration can specify only one calling convention. Valid calling conventions include: OS, COBOL, PLI, FORTRAN

User Response: Remove extra calling convention specifiers.

CCN3367 Only functions or typedefs of functions can be given a calling convention.

Explanation: A calling convention protocol keyword has been applied to an identifier that is not a function type or a typedef to a function type.

User Response: Check that correct identifier is specified or remove #pragma.

CCN3369 The function cannot be redeclared with a different calling convention.

Explanation: The redeclaration of this function cannot have a different calling convention than the previous declaration. The function could have been given a calling convention through a typedef, or via a previous declaration.

User Response: Make sure all declarations of the function specify the same calling convention.

CCN3374 Pointer types ″&1″ and ″&2″ are not compatible.

Explanation: The types pointed to by the two pointers are not compatible.

User Response: Change the types to be compatible.

CCN3376 Redeclaration of &1 has a different number of fixed parameters than the previous declaration.

Explanation: The number of fixed parameters in the redeclaration of the function does not match the original number of fixed parameters.

User Response: Change the declarations to have the same number of parameters, or rename or remove one of the declarations.

CCN3377 The type ″&1″ of parameter &2 differs from the previous type ″&3″.

Explanation: The type of the corresponding parameter in the previous function declaration is not compatible.

User Response: Change the parameter declaration or rename the function declaration.

CCN3378 Prototype for function &1 cannot contain ″...″ when mixed with a nonprototype declaration.

Explanation: A function prototype and a nonprototype declaration can not be compatible if one contains ″...″.

User Response: Convert nonprototype declaration to a prototyped one or remove the ″...″.

CCN3379 Prototype for function &1 must contain only promoted types if prototype and nonprototype declarations are mixed.

Explanation: Nonprototype declarations have their parameters automatically promoted. Integral widening conversions are applied to integral types and float is converted into double.

User Response: Promote the parameter types in the prototyped declaration.

CCN3380 Parameter &1 has type ″&2″ which promotes to ″&3″.

Explanation: Nonprototype declarations have their parameters automatically promoted. Integral widening conversions are applied to integral types and float is converted into double.

User Response: Promote the parameter types in the prototyped declaration.

CCN3381 The type ″&1″ of parameter &2 in the prototype declaration is not compatible with the corresponding parameter type ″&3″ in the nonprototype declaration.

Explanation: The types of the parameters must be compatible.

User Response: Change the parameters so that they are compatible.

CCN3382 The type ″&1″ of identifier &2 differs from previous type ″&3″.

Explanation: The two types are not compatible.

User Response: Change the parameter types so that they are compatible.

CCN3383 Expecting ″&1″ to be an external identifier.

Explanation: The identifier must have external linkage.

User Response: Change the storage class to extern.

CCN3384 Expecting ″&1″ to be a function name.

Explanation: ″&1″ should be a function symbol.

User Response: Specify a different name or change the type of the symbol.

CCN3387 The enum cannot be packed to the requested size. Change the enumeration value or change the #pragma enum().

Explanation: Enums may be 1, 2, or 4 bytes in size.

User Response: Change the enumeration value or change the #pragma enum().

CCN3388 Value &1 specified in #pragma &2 is out of range.

Explanation: Refer to the C/C++ Language Reference for more information about the valid values for the #pragmas.

User Response: Specify a different value.

CCN3389 Some program text not scanned due to &1 option or #pragma &2.

Explanation: MARGINS or SEQUENCE option, or #pragma margins or sequence was used to limit the valid text region in a source file.

User Response: Remove the MARGINS or SEQUENCE option, or remove the #pragma margins or sequence, or specify a more inclusive text region.

CCN3390 The function or variable &1 cannot be declared as an import in the same compilation unit in which it is defined.

Explanation: An object or function has both a definition and an import directive in this compilation unit. This creates a conflict, since the function or object can be defined either here or where it is exported from, but not both.

User Response: Remove the #pragma import directive or __import keyword or change the definition of the object or function into an extern declaration.

CCN3393 &1 value must contain only decimal digits.

Explanation: A non-numeric character was encountered in the &1 value.

User Response: Check the syntax of the value given.

CCN3394 Ordinal value on #pragma &1 is out of range.

Explanation: The specified ordinal number should be between 0 and 65535, inclusive.

User Response: Change the value accordingly.

CCN3395 Variable &1 must be an external object or a function name for use with #pragma import.

Explanation: The identifier specified by the pragma is not a function or external object.

User Response: Declare the object with storage class ″extern″.

CCN3396 Option &1 is incompatible with option &2 and is ignored.

Explanation: The option is not compatible with another option so it is ignored.

User Response: Remove one of the options.

CCN3397 Undefined function or variable &1 cannot have a #pragma export.

Explanation: Only defined variables or functions can be specified as an export.

User Response: Define the function or variable.

CCN3398 Bit-field type specified for &1 is non-portable. The type should be signed int, unsigned int or int.

Explanation: The specification of the bit-field type may cause problems with porting the code to another system.

User Response: Change the type specifier.

CCN3399 The alignment of a structure/union is determined at the left brace of the definition.

Explanation: The alignment of an aggregate is constant throughout its definition.

CCN3400 #pragma &1 must appear only once in any C file.

User Response: Remove all but one of the specified #pragma directives.

CCN3401 Function &1 must be defined for #pragma entry.

Explanation: The function must be defined for it to be specified using #pragma entry.

User Response: Define the function.

CCN3402 &1 must be an externally-defined function for use with #pragma entry.

Explanation: The identifier must be defined as a function with external linkage for it to be specified using #pragma entry.

User Response: Define the function.

CCN3408 The linkage protocol is not supported on the target platform.

Explanation: An attempt to use an unsupported linkage protocol was made.

User Response: Remove the linkage protocol keywords.

CCN3409 The static variable ’&1’ is defined but never referenced.

Explanation: A variable that is defined but never used probably serves no purpose.

User Response: Remove the variable definition if you are not going to use the variable.

CCN3410 The automatic variable ’&1’ is defined but never referenced.

Explanation: A variable that is defined but never used likely serves no purpose.

User Response: Remove the variable definition.

CCN3411 An array that is not an lvalue cannot be subscripted.

Explanation: A non-lvalue array is created when a function returns a structure that contains an array. This array cannot be dereferenced.

User Response: Remove the subscript.

CCN3412 Referenced variable '&1', which was not initialized in its declaration.

Explanation: The variable referenced was not initialized in its declaration. At the point of the first reference, the variable might or might not have already been set to a value, depending on the code executed prior to the point of the first reference.

User Response: This is an informational message to aid debugging. Either initialize the variable in its declaration, or trace the code carefully to make sure that it is set to a value prior to the first reference.

CCN3413 A goto statement is used.

Explanation: The use of goto statements may result in code that is more difficult to trace.

User Response: Replace the goto statement with equivalent structured-programming constructs.

CCN3414 The parameter '&1' is never referenced.

Explanation: The parameter is passed to the function, but is not referenced anywhere within the function body.

User Response: Remove the parameter from the function prototype.

CCN3415 The external function definition '&1' is never referenced.

Explanation: A function that is defined but never used likely serves no purpose.

User Response: Remove the function definition, unless needed in another compilation unit.

CCN3416 Taking the negative of the most negative value, '&1', of a signed type will cause truncation.

Explanation: The negative of the most negative value cannot be represented as a positive value of the same type.

User Response: Change the value or use a larger data type.

CCN3417 The function &1 is not defined but has #pragma inline directive specified.

Explanation: A #pragma inline has been applied to an identifier which does not exist or does not correspond to a function.

User Response: Check that correct identifier is specified or remove #pragma.

CCN3418 '&1' does not evaluate to a constant that fits in its signed type.

Explanation: The expression evaluates to a number that is not within the range that can be stored by the target.

User Response: Change the expression so it evaluates to a value in the valid range.

CCN3419 Converting &1 to type ″&2″ does not preserve its value.

Explanation: The user cast converts &1 to a type that cannot contain the value of the original type.

User Response: Change the cast.

CCN3420 An unsigned comparison is performed between an unsigned value and a negative constant.

Explanation: Comparing an unsigned value with a signed value may produce unexpected results.

User Response: Type-cast the unsigned value to a signed type if a signed comparison is desired, or type-cast the negative constant to an unsigned type if an unsigned comparison is desired.

CCN3421 The comparison is always true.

Explanation: The type specifiers of the values being compared result in a constant result.

User Response: Simplify or remove the conditional expression.

CCN3422 The comparison is always false.

Explanation: The type specifiers of the values being compared result in a constant result.

User Response: Simplify or remove the conditional expression.

CCN3423 The comparison may be rewritten as '&1'.

Explanation: The type specifiers of the values being compared may allow the expression to be simplified.

User Response: Simplify the comparison expression.

CCN3424 The condition is always true.

Explanation: Because the value of the conditional expression is constant, it may be possible to simplify or remove the conditional test.

User Response: Change the conditional expression or remove the conditional test.

CCN3425 The condition is always false.

Explanation: Because the value of the conditional expression is constant, it may be possible to simplify or remove the conditional test.

User Response: Change the conditional expression or remove the conditional test.

CCN3426 An assignment expression is used as a condition. An equality comparison (==) may have been intended.

Explanation: A single equal sign '=' is often mistakenly used as an equality comparison operator.

User Response: Ensure an assignment operation was intended.

CCN3427 A constant expression is used as a switch condition.

Explanation: The same code path will be taken through every execution of the switch statement.

User Response: Change the switch expression to be a non-constant value or remove the unused portions of the switch structure.

CCN3428 The left-hand side of a shift expression is an unparenthesized arithmetic expression which has a higher precedence.

Explanation: The left-hand expression is evaluated before the shift operator.

User Response: Place parentheses around the left-hand expression to make the order of operations explicit.

CCN3429 The right-hand side of a shift expression is an unparenthesized arithmetic expression which has a higher precedence.

Explanation: The right-hand expression is evaluated before the shift operator.

User Response: Place parentheses around the right-hand expression to make the order of operations explicit.

CCN3430 The result of a comparison is either 0 or 1, and may not be appropriate as operand for another comparison operation.

Explanation: The comparison expression may be malformed.

User Response: Ensure that the resulting value from the comparison is appropriate for use in the following comparison.

CCN3431 The left-hand side of a bitwise &&, |, or ˆ expression is an unparenthesized relational, shift, or arithmetic expression which has a higher precedence.

Explanation: The left-hand expression is evaluated before the bitwise operator.

User Response: Place parentheses around the left-hand expression to make the order of operations explicit.

CCN3432 The right-hand side of a bitwise &&, |, or ˆ expression is an unparenthesized relational, shift, or arithmetic expression which has a higher precedence.

Explanation: The right-hand expression is evaluated before the bitwise operator.

User Response: Place parentheses around the right-hand expression to make the order of operations explicit.

CCN3433 The right-hand side of a bitwise shift expression should be positive and less than the width in bits of the promoted left operand.

Explanation: This expression may not be portable.

User Response: Change the shift expression.

CCN3434 The left-hand side of a bitwise right shift expression has a signed promoted type.

Explanation: This expression may not be portable.

User Response: Change the shift expression.

CCN3435 An expression statement should have some side effects because its value is discarded.

Explanation: If an expression statement has no side effects, then it may be possible to remove the statement with no change in program behaviour.

User Response: Change or remove the expression statement.

CCN3436 Left-hand side of comma expression should have side effects because its value is discarded.

Explanation: A comma expression evaluates to its right-hand operand.

User Response: Change the expression.

CCN3437 The init or re-init expression of a for statement should have some side effects since its value is discarded.

Explanation: If the init and/or the re-init expression of a for statement have no side effects, the loop may not execute as desired.

User Response: Change the init and/or re-init expressions.

CCN3438 The value of the variable ’&1’ may be used before being set.

Explanation: Because the variable has not been initialized, its value is undefined. The results of using an undefined variable are unpredictable.

User Response: Add an initialization statement or change the expression.

CCN3439 Assigning enum type ’&1’ to enum type ’&2’ may not be correct.

Explanation: The values of the enumerated types may be incompatible.

User Response: Change the types of the values being assigned.

CCN3440 Cannot assign an invalid enumerator value to enum type ’&1’.

Explanation: The value being assigned is not a member of the enumeration.

User Response: Change the value being assigned, or make it an enumeration member.

CCN3441 The macro definition will override the keyword ’&1’.

Explanation: Overriding a C keyword with a preprocessor macro may cause unexpected results.

User Response: Change the name of the macro or remove it.

CCN3442 A trigraph sequence occurs in a character literal.

Explanation: The trigraph sequence will be converted. A literal interpretation may have been desired.

User Response: Change the value of the character literal.

CCN3443 A trigraph sequence occurs in a string literal.

Explanation: The trigraph sequence will be converted. A literal interpretation may have been desired.

User Response: Change the value of the string literal.

CCN3444 The opening brace is redundant.

Explanation: The initialization expression contains extra, possibly unnecessary, braces.

User Response: Remove the extra braces.

CCN3445 The closing brace is redundant.

Explanation: The initialization expression contains extra, possibly unnecessary, braces.

User Response: Remove the extra braces.

CCN3446 Array element(s) [&1] will be initialized with a default value of 0.

Explanation: Some array elements were not explicitly initialized. They will be assigned the default value.

User Response: Add initializations if necessary.

CCN3447 The member(s) starting from ’&1’ will be initialized with a default value of 0.

Explanation: Some members were not explicitly initialized. They will be assigned the default value.

User Response: Add initializations if necessary.

CCN3448 Assigning a packed struct to an unpacked struct, or vice versa, requires remapping.

Explanation: Assignments between packed/unpacked structures may produce incorrect results.

User Response: Change the type qualifiers of the values in the assignment.

CCN3449 Missing return expression.

Explanation: If a function has a non-void return type, then all return statements must have a return expression of the correct type.

User Response: Add a return expression.

CCN3450 Obsolete non-prototype-style function declaration.

Explanation: The K&R-style function declaration is obsolete.

User Response: Change the function declaration to the prototyped style.

CCN3451 The target integral type cannot hold all possible values of the source integral type.

Explanation: Data loss or truncation may occur because of the type conversions.

User Response: Change the types of the values in the expression.

CCN3452 Assigning a floating point type to an integral type may result in truncation.

Explanation: Data loss or truncation may occur because of the type conversions.

User Response: Change the types of the values in the expression.

CCN3453 Assigning a floating point type to another floating point type with less precision.

Explanation: Data loss or truncation may occur because of the type conversions.

User Response: Change the types of the values in the expression.

CCN3454 &1 condition evaluates to &2.

Explanation: This message traces preprocessor expression evaluation.

User Response: No response required.

CCN3455 defined(&1) evaluates to &2.

Explanation: This message traces preprocessor #ifdef and #ifndef evaluation.

User Response: No response required.

CCN3456 Stop skipping tokens.

Explanation: This messages traces conditional compilation activity.

User Response: No response required.

CCN3457 File &1 has already been included.

Explanation: This #include directive is redundant.

User Response: Remove the #include directive.

CCN3458 #line directive changing line to &1 and file to &2.

Explanation: This message traces #line directive evaluation.

User Response: No response required.

CCN3459 #line directive changing line to &1.

Explanation: This message traces #line directive evaluation.

User Response: No response required.

CCN3460 &1 nesting level is &2.

Explanation: This message traces conditional compilation activity.

User Response: No response required.

CCN3461 Generating precompiled header file &1.

Explanation: This message traces precompiled header generation activity.

User Response: No response required.

CCN3462 Precompiled header file &1 is found but not used because it is not up to date.

Explanation: This message traces precompiled header file generation activity.

User Response: No response required.

CCN3463 Using precompiled header file &1.

Explanation: This message traces precompiled header file generation activity.

User Response: No response required.

CCN3464 Begin skipping tokens.

Explanation: This messages traces conditional compilation activity.

User Response: No response required.

CCN3465 #undef undefining macro name &1.

Explanation: This message traces #undef preprocessor directive evaluation.

User Response: No response required.

CCN3466 Unary minus applied to an unsigned type.

Explanation: The negation operator is inappropriate for unsigned types.

User Response: Remove the operator or change the type of the operand.

CCN3467 String literals concatenated.

Explanation: Two string literals, each delimited by quotation marks, have been combined into a single literal.

User Response: No response is necessary. This is an informational message.

CCN3468 Macro name &1 on #define is also an identifier.

Explanation: The name of the macro has already been used.

User Response: Change the name of the macro.

CCN3469 The static function '&1' is declared or defined but never referenced.

Explanation: A function that is defined but never used serves no purpose.

User Response: Remove the function definition.

CCN3470 Function 'main' should return int, not void.

Explanation: According to the ANSI/ISO standard, main should return int not void. Earlier standards (such as k&R) allowed a void return type for main.

User Response: Change the return type of the function.

CCN3471 Case label is not a member of enum type '&1'

Explanation: Case labels must be members of the type of the switch expression.

User Response: Change the value of the case label.

CCN3472 Statement is unreachable.

Explanation: The flow of execution causes this statement to never be reached.

User Response: Change the control flow in the program, or remove the unreachable statement.

CCN3473 An unintended semi-colon may have created an empty loop body.

Explanation: The loop body has no statements, and the conditional expression has no side effects.

User Response: If this is what was intended, use '{}' instead of a semi-colon as empty loop body to avoid this message.

CCN3474 Loop may be infinite.

Explanation: The value of the conditional expression and/or the lack of exit points may result in an infinite loop.

User Response: Adjust the conditional expression or add loop exit statements.

CCN3475 The real constant arithmetic expression folds to positive infinity.

Explanation: Constant folding results in an overflow.

User Response: Change the expression.

CCN3476 The real constant arithmetic expression folds to negative infinity.

Explanation: Constant folding results in an overflow.

User Response: Change the expression.

CCN3478 The then branch of conditional is an empty statement.

Explanation: If the condition is true, then no statement is executed.

User Response: Add a statement to be executed, or remove the conditional statement.

CCN3479 Both branches of conditional statement are empty statements.

Explanation: A conditional statement with empty branches is possibly degenerate.

User Response: Add code to the conditional branches.

CCN3480 Missing break statement allows fall-through to this case.

Explanation: The preceding case did not end with a break, return, or goto statement, allowing the path of execution to fall-through to the code in this case.

User Response: Add an appropriate terminating statement to the previous case, unless the fall-through was intentional.

CCN3481 The end of the function may be reached without returning a value.

Explanation: A return statement should be used to exit any function whose return type is non-void.

User Response: Add a return statement, or change the function to return void.

CCN3482 The opening brace before this point is redundant.

Explanation: The initialization expression contains extra, possibly unnecessary, braces.

User Response: Remove the extra braces.

CCN3483 Switch statement contains no cases or only default case.

Explanation: Code within a switch statement block that is not preceded by either 'default' or 'case' is never executed, and may be removed. Switch statements with neither 'default' or 'case' are probably incorrect.

User Response: Change the switch statement to include cases.

CCN3484 External name &1 has been truncated to &2.

Explanation: The external name exceeds the maximum length and has been truncated. This may result in unexpected behavior if two different names become the same after truncation.

User Response: Reduce the length of the external name.

CCN3485 Parameter declaration list is incompatible with declarator for &1.

Explanation: An attempt has been made to attach a parameter declaration list with a declarator which cannot have one.

User Response: Change declarator or remove parameter declaration list.

CCN3486 A pointer to an incomplete type cannot be indexed.

Explanation: An index has been used with a pointer to an incomplete type.

User Response: Declare the type that is pointed at or remove the index.

CCN3487 An argument cannot be an incomplete struct or union.

Explanation: An incomplete aggregate cannot be used as an argument to a function.

User Response: Declare the type that is pointed at or use a pointer to the aggregate.

CCN3489 The incomplete struct or union tag &1 was not completed before going out of scope.

Explanation: A struct or union tag was declared inside a parameter list or a function body, but no member declaration list was provided.

User Response: If the struct or union tag was declared inside a parameter list, provide a member declaration list at file scope. If the tag was declared inside a function body, provide a member declaration list within that function body.

CCN3490 The static variable '&1' is set but never referenced.

Explanation: A variable that is initialized but never used serves no purpose.

User Response: Remove the variable definition if you do not intend to use it.

CCN3491 The automatic variable '&1' is set but never referenced.

Explanation: A variable that is initialized but never used likely serves no purpose.

User Response: Remove the variable definition if you do not intend to use it.

CCN3492 Redefinition of &1 hides previous definition.

Explanation: The definition within the current scope hides a definition with the same name in an enclosing scope.

User Response: Change the name to avoid redefining it.

CCN3493 The external variable '&1' is defined but never referenced.

Explanation: A variable that is defined but never used likely serves no purpose.

User Response: Remove the variable definition, unless needed in another compilation unit.

CCN3494 The external variable '&1' is set but never referenced.

Explanation: A variable that is initialized but never used serves no purpose.

User Response: Remove the variable definition, unless needed in another compilation unit.

CCN3495 Pointer type conversion found.

Explanation: An attempt is being made to convert a pointer of one type to a pointer of another type.

User Response: Check the types of the values involved in the expression, and make them compatible.

CCN3496 Parameter(s) for #pragma &1 are of the wrong type.

Explanation: The parameter for the pragma is incorrect and of the wrong type.

User Response: Look up correct type in the C/C++ Language Reference.

CCN3497 Incomplete enum type not allowed.

Explanation: An incomplete enum is being used where a complete enum type is required.

User Response: Complete the type declaration.

CCN3498 Member of struct or union cannot be incomplete type.

Explanation: An incomplete aggregate is being used where a complete struct or union is required.

User Response: Complete the type declaration.

CCN3499 Function 'main' should return int.

Explanation: A return type other than int was specified for function main.

User Response: Change the return type to int.

CCN3503 Option ″&1″ is not supported for &2.

Explanation: The option specified is not supported on this operating system.

User Response: Remove the option.

CCN3505 Type ″&1″ of identifier ″&2″ was incomplete at the end of its scope.

Explanation: A incomplete declaration was made of some identifier and it is still incomplete at the end of its scope.

User Response: Complete the declaration.

CCN3508 Option &1 for #pragma &2 is not supported.

Explanation: For a list of all valid options for #pragma directives, see the C/C++ Language Reference.

User Response: Ensure the #pragma syntax and options are correct.

CCN3509 Symbol &1 on a #pragma &2 was not found.

Explanation: For a list of all valid options for #pragma directives, see the C/C++ Language Reference.

User Response: Ensure the #pragma syntax and options are correct.

CCN3512 An initializer is not allowed for ″&1″.

Explanation: An attempt was made to initialize an identifier whose type does not permit initialization.

User Response: Remove the initializer.

CCN3513 Array element designator exceeds the array dimension. Designator will be ignored.

Explanation: The value of the designator was larger than the dimension declared for the array object.

User Response: Change the expression forming the array index.

CCN3514 Array element designator cannot be applied to an object of type ″&1″.

Explanation: An array element designator can only be applied to an object of array type.

User Response: Remove subscript.

CCN3515 Member designator cannot be applied to an object of type ″&1″.

Explanation: A member designator can only be applied to an object of type struct or union.

User Response: Remove member designator.

CCN3517 Option &1 for #pragma is not supported.

Explanation: For a list of all valid options for #pragma directives, see the C/C++ Language Reference.

User Response: Ensure the #pragma syntax and options are correct.

CCN3518 Option(s) for #pragma &1 are missing or incorrectly specified.

Explanation: #pragma &1 is not correctly specified.

User Response: Ensure the #pragma syntax and options are correct.

CCN3519 Index operator ([]) cannot be applied to pointer to void.

Explanation: Index operator ([]) can only be applied to arrays or pointers to objects.

User Response: Change the operand.

CCN3520 Switch block begins with declarations or unlabeled statements that are unreachable.

Explanation: Code within a switch block must be labeled with either ’case’ or ’default’ to be reachable.

User Response: Add a label or remove the unreachable code.

CCN3521 Pointer arithmetic can only be applied to a arrays that are lvalues.

Explanation: Because the array is compiler-generated, it is not an lvalue. Therefore, you cannot apply pointer arithmetic to it.

User Response: Change the expression.

CCN3522 Unable to open precompiled header &1 for output.

Explanation: The compiler was unable to open the precompiled header file.

User Response: Ensure that the compiler has write access to the precompiled header files.

CCN3524 The _Packed qualifier can only qualify a struct or union.

Explanation: The _Packed qualifier is only valid for structures and unions.

User Response: Remove _Packed qualifier.

CCN3531 End of precompiled header processing.

Explanation: The compiler has finished processing a precompiled header.

User Response: No response required. This message merely traces the activity of the precompiled header processing.

CCN3532 Macro ″&1″ is required by the precompiled header and is defined differently than when the precompiled header was created.

Explanation: The referenced macro was expanded during the creation of the precompiled header and is now defined differently. This prevents the precompiled header from being used for this compilation.

User Response: If necessary, redefine the macro, or regenerate the precompiled header using the new macro definition.

CCN3533 One or more assertions are defined that were not defined when the precompiled header was created.

Explanation: An assertion is defined that was not defined when the precompiled header was generated. Because the effect of the new assertion is unknown, the precompiled header cannot be used for this compilation.

User Response: Do not define the assertion, or regenerate the precompiled header with the new assertion.

CCN3534 One or more macros are defined that were not defined when the precompiled header was created.

Explanation: A macro is defined that was not defined when the precompiled header was generated. Because the effect of the new macro is unknown, the precompiled header cannot be used for this compilation.

User Response: Do not define the macro or regenerate the precompiled header with the new macro.

CCN3535 Compiler options do not match those in effect when the precompiled header was created.

Explanation: The compiler options in use are not compatible with those used when the precompiled header was generated. The precompiled header cannot be used.

User Response: Use the same options as when the precompiled header was generated or regenerate the precompiled header with the new options.

CCN3536 Assertion ″&1″ is required by the precompiled header and is not defined.

Explanation: The referenced assertion was tested during the creation of the precompiled header and is not defined. This prevents the precompiled header from being used for this compilation.

User Response: If necessary, redefine the assertion, or regenerate the precompiled header without the assertion.

CCN3537 Macro ″&1″ is required by the precompiled header and is not defined.

Explanation: The referenced macro was expanded during the creation of the precompiled header and is not defined. This prevents the precompiled header from being used for this compilation.

User Response: If necessary, redefine the macro, or regenerate the precompiled header without the macro.

CCN3538 Unable to use precompiled header &1.

Explanation: The precompiled header cannot be used for this compilation. A subsequent message will explain the reason.

User Response: Correct the problem indicated by the subsequent message.

CCN3539 Expecting &1 and found &2.

Explanation: The header file being included is not the next header in the sequence used to generate the precompiled header. The precompiled header cannot be used for this compilation.

User Response: #include the correct header or regenerate the precompiled header using the new sequence of #include directives.

CCN3545 The decimal size is outside the range of 1 to &1.

Explanation: The specified decimal size should be between 1 and DEC_DIG.

User Response: Specify the decimal size between 1 and DEC_DIG.

CCN3546 The decimal precision is outside the range of 0 to &1.

Explanation: The specified decimal precision should be between 0 and DEC_PRECISION.

User Response: Specify the decimal precision between 0 and DEC_PRECISION.

CCN3547 The decimal size is not valid.

Explanation: The decimal size must be a positive constant integral expression.

User Response: Specify the decimal size as a positive constant integral expression.

CCN3548 The decimal precision is not valid.

Explanation: The decimal precision must be a constant integral expression.

User Response: Specify the decimal precision as a constant integral expression.

CCN3549 The decimal precision is bigger than the decimal size.

Explanation: The specified decimal precision should be less than or equal to the decimal size.

User Response: Specify the decimal precision less than or equal to the decimal size.

CCN3550 The decimal constant is out of range.

Explanation: The compiler detected a decimal overflow in scanning a decimal constant.

User Response: Change the decimal constant so that it does not exceed the maximum value.

CCN3551 The fraction part of the result was truncated.

Explanation: Due to limitations on the number of digits representable, the calculated intermediate result may result in truncation in the decimal places after the operation is performed.

User Response: Check to make sure that no significant digit is lost.

CCN3552 The pre- and post- increment and decrement operators cannot be applied to type &1.

Explanation: The decimal types with no integral part cannot be incremented or decremented.

User Response: Reserve at least one digit in the integral part of the decimal types.

CCN3553 Only decimal types can be used with the &1 operator.

Explanation: The operand of the digitsof or precisionof operator is not valid. The digitsof and precisionof operators can only be applied to decimal types.

User Response: Change the operand.

CCN3554 Whole-number-part digits in the result may have been lost.

Explanation: Due to limitations on the number of digits representable, the calculated intermediate result may result in loss of digits in the integer portion after the operation is performed.

User Response: Check to make sure that no significant digit is lost.

CCN3555 Digits have been lost in the whole-number part.

Explanation: In performing the operation, some non-zero digits in the whole-number part of the result are lost.

CCN3556 Digits may have been lost in the whole-number part.

Explanation: In performing the operation, some digits in the whole-number part of the result may have been lost.

User Response: Check to make sure that no significant digit is lost.

CCN3557 The name in option &1 is not valid. The option is reset to &2.

Explanation: The name specified as a suboption of the option is syntactically or semantically incorrect and thus can not be used.

User Response: Make sure that the suboption represents a valid name. For example, in option LOCALE(localename), the suboption 'localename' must be a valid locale name which exists and can be used. If not, the LOCALE option is reset to NOLOCALE.

CCN3558 #pragma &1 is ignored because the locale compiler option is not specified.

Explanation: The locale compiler option is required for #pragma &1

User Response: Remove all the #pragma &1 directives or specify the locale compiler option.

CCN3559 #pragma filetag is ignored because the conversion table from &1 to &2 cannot be opened.

Explanation: During compilation, source code is converted from the code set specified by #pragma filetag to the code set specified by the locale compiler option, if they are different. A conversion table form &1 to &2 must be loaded prior to the conversion. No conversion is done when the conversion table is not found.

User Response: Create the conversion table from &1 to &2 and ensure it is accessible from the compiler. If message files are used in the application to read and write data, a conversion table from &2 to &1 must also be created to convert data from runtime locale to the compile time locale.

CCN3560 Error messages are not converted because the conversion table from &1 to &2 cannot be opened.

Explanation: Error messages issued by C/370 are written in code page 1047. These messages must be converted to the code set specified by the locale compiler option because they may contain variant characters, such as #. Before doing the conversion, a conversion table from &1 to &2 must be loaded. The error messages are not converted because the conversion table cannot be found.

User Response: Make sure the conversion table from &1 to &2 is accessible from the compiler.

CCN3561 No conversion on character &1 because it does not belong to the input code set &2.

Explanation: No conversion has be done for the character because it does not belong to the input code set.

User Response: Remove or change the character to the appropriate character in the input code set.

CCN3562 Incomplete character or shift sequence was encountered during the conversion of the source line.

Explanation: Conversion stops because an incomplete character or shift sequence was encountered at the end of the source line.

User Response: Remove or complete the incomplete character or shift sequence at the end of the source line.

CCN3563 Only conversion table that map single byte characters to single byte characters is supported.

Explanation: Compiler is expected single byte to single byte character mapping during conversion. Conversion stops when there is insufficient space in the conversion buffer.

User Response: Make sure the conversion table is in single byte to single byte mapping.

CCN3564 Invalid conversion descriptor was encountered during the conversion of the source line.

Explanation: No conversion was performed because conversion descriptor is not valid.

CCN3565 #pragma &1 must appear on the first directive before any C code.

Where: &1 pragma type *CHAR 100

User Response: Put this #pragma as the first directive before any C code.

CCN3566 Option DECK ignored because option OBJECT specified.

Explanation: The second option must not be specified for the first to have an effect.

User Response: Remove the first or second option.

CCN3567 Option OFFSET ignored because option LIST not specified.

Explanation: The second option must be specified for the first to have an effect.

User Response: Specify the second option, or remove the first.

CCN3568 The external name &1 in #pragma csect conflicts with another csect name.

Explanation: A #pragma csect was specified with a name which has already been specified as a csect name.

User Response: Ensure that the two csect names are unique.

CCN3569 A duplicate #pragma csect(&1) is ignored.

Explanation: Only one #pragma csect may be specified for either CODE or STATIC.

User Response: Remove the duplicate #pragma csect.

CCN3570 The #pragma map name &1 must not conflict with a #pragma csect name or the csect name generated by the compiler.

Explanation: The external name used in the #pragma map is identical to the external name specified on the #pragma csect or the name generated by the compiler.

User Response: Change the name on the #pragma csect or turn off the CSECT option.

CCN3571 The external name &1 must not conflict with the name in #pragma csect or the csect name generated by the compiler.

Explanation: The external name specified is identical to the name specified on a #pragma csect or the name generated by the CSECT option.

User Response: Change the name on the #pragma csect or turn off the CSECT option.

CCN3572 Expected text &1 was not encountered on option &2.

User Response: Use the correct syntax for specifying the option

CCN3573 To use the builtin form of the &1 function add the #include <&2> directive.

User Response: Add the specified #include in order to optimize code.

CCN3574 Unable to open event file &1.

Explanation: The compiler was unable to open the event file.

User Response: Ensure that there is enough disk space.

CCN3575 Csect option is ignored due to naming error.

Explanation: The compiler was unable to generate valid csect names.

User Response: Use #pragma csect to name the code and static control sections.

CCN3576 Csect name &1 has been truncated to &2.

Explanation: The static, data and test csect names have been truncated to 8 characters.

CCN3577 Obsolete option OPTIMIZE(2) defaults to OPTIMIZE(1).

Explanation: Optimize(2) is no longer supported and has been defaulted to 1.

CCN3578 The csect name &1 must not conflict with a csect name generated by the compiler.

Explanation: The code and static csect names are identical. Either the compiler is unable to generate unique names or a #pragma csect is using a duplicate name.

User Response: Use #pragma csect to name the code and static control sections.

CCN3585 Obsolete option HWOPTS defaults to corresponding ARCHITECTURE option.

Explanation: HWOPTS is no longer supported and has been replaced by ARCHITECTURE.

User Response: Use the ARCHITECTURE option to take advantage of hardware.

CCN3586 Test csect name &1 has been truncated to &2.

Explanation: The compiler generated test csect name has been truncated to 8 characters.

User Response: Use the CSECT() option to allow test csect names longer than 8 chars.

CCN3600 3600 - 3631 are LE messages.

Explanation: Refer to the LE manuals for further information about these messages

CCN3671 The header file name in the #include directive cannot be empty.

User Response: Specify a non-empty header file name in the #include directive.

CCN3675 The return type is not valid for a function of this linkage type

Explanation: The linkage type of the function puts certain restrictions on the return type, on which the function definition violated.

User Response: Check the linkage type restrictions and change the return type.

CCN3676 Function ″&1″ which returns a return code cannot be defined.

Explanation: The function has FORTRAN linkage type with the RETURNCODE option. Therefore it should be a FORTRAN function defined somewhere else and referenced here (should not be defined in the compile unit).

User Response: Make sure the function is a FORTRAN function.

CCN3677 Option LONGNAME is turned on because option DLL is specified.

Explanation: Option LONGNAME is turned on by the compiler because DLL option is specified.

CCN3678 Option RENT is turned on because option DLL is specified.

Explanation: Option RENT is turned on by the compiler because DLL option is specified.

CCN3679 Option LONGNAME is turned on because option EXPORTALL is specified.

Explanation: Option LONGNAME is turned on by the compiler because EXPORTALL option is specified.

CCN3680 Option RENT is turned on because option EXPORTALL is specified.

Explanation: Option RENT is turned on by the compiler because EXPORTALL option is specified.

CCN3681 #pragma export(&1) is ignored; both LONGNAME and RENT options must be specified.

Explanation: The variable/function is not exported because both LONGNAME and RENT must be specified to export functions/variables.

User Response: Make sure both LONGNAME and RENT options are specified.

CCN3682 ″&1″ will not be exported because #pragma variable(&2,NORENT) is specified.

Explanation: Variables with NORENT option cannot be exported.

CCN3683 ″&1″ will not be exported because it does not have external storage class.

Explanation: Only objects with external storage class can be exported.

CCN3684 Exporting function main is not allowed.

Explanation: Main cannot be exported.

User Response: Remove the pragma export for main.

CCN3685 ″&1″ will not be exported because it is not external defined.

Explanation: The variable cannot be exported because it is not defined here.

User Response: Remove the pragma export for the variable.

CCN3686 Unexpected keyword(s). One or more keywords were found in an invalid location.

Explanation: One or more keywords were found in an invalid location.

User Response: Remove the keyword(s) or place them immediately to the left of the identifier to which they apply.

CCN3687 The &1 keyword cannot be applied to the return type of a function.

Explanation: The keyword is being applied to the return type of a function.

User Response: Remove the keyword.

CCN3688 Declaration cannot specify conflicting keywords &1 and &2.

Explanation: The keywords conflict and cannot both be used in the same declaration.

User Response: Remove one of the keywords.

CCN3689 The &1 keyword was specified more than once in the declaration.

Explanation: The keyword was used more than once in the same declaration.

User Response: Remove one of the keywords.

CCN3690 Builtin function &1 is unrecognized. The default linkage convention is used.

Explanation: The function specified in the pragma linkage builtin is not a builtin function.

User Response: Check the function name and correct; or remove the pragma if it is not a builtin function.

CCN3691 The &1 keyword can only be applied to functions.

Explanation: The keyword has been applied to an identifier which does not correspond to a function type.

User Response: Check that the correct identifier is specified or remove the keyword.

CCN3692 Both ″main″ and ″WinMain″ are defined in this compilation unit. Only one of them is allowed.

Explanation: In each compilation unit, only one of ″main″ and ″WinMain″ is allowed.

User Response: Remove either ″main″ or ″WinMain″.

CCN3693 The &1 keyword conflicts with a previously specified keyword.

Explanation: The keyword conflicts with another keyword specified in the same declaration.

User Response: Remove one of the keywords.

CCN3694 Option LONGNAME is turned on because a qualifier is specified on the CSECT option.

Explanation: Option LONGNAME is turned on by the compiler when the CSECT option is specified with a qualifier.

CCN3695 #pragma export(&1) is ignored; LONGNAME option must be specified.

Explanation: The variable/function is not exported because LONGNAME must be specified to export functions/variables.

User Response: Make sure LONGNAME option is specified.

CCN3708 Only functions or typedefs of functions can be specified on #pragma linkage directive.

Explanation: The name specified on #pragma linkage is not a function.

User Response: Check for typo errors; remove the #pragma linkage.

CCN3709 Structure members cannot follow zero-sized array.

Explanation: The zero-sized array must be the last member in the structure.

User Response: Remove members that occur after the zero-sized array.

CCN3710 Option &1 ignored because option &2 specified.

CCN3711 Option &1 ignored.

CCN3712 Duplicate function specifier ″&1″ ignored.

CCN3713 Keyword ″&1″ is not allowed.

CCN3714 #include searching for file &1.

CCN3715 Storage class &1 cannot be used for structure members.

Explanation: The storage class is not appropriate for this declaration. Restrictions include: 1) Storage class specifier not allowed on aggregate members, casts, sizeof or offsetof declarations. 2) Declarations at file scope cannot have ’register’ or ’auto’ storage class.

User Response: Specify a different storage class.

CCN3717 Only external data and functions can be declared as export or import.

Explanation: Either the _Export or _Import keyword, or #pragma export or #pragma import was used with data or a function which is not external.

CCN3721 The ″&1″ qualifier is not supported on the target platform.

Explanation: The specified qualifier is not supported on the target platform and will have no effect.

CCN3722 #pragma linkage &1 ignored for function &2.

Explanation: A conflicting linkage type, or a #pragma environment, has been specified for this function.

User Response: Check what has been specified before and remove the conflicts.

CCN3723 #pragma environment is ignored because function &1 already has linkage type &2.

Explanation: A pragma linkage has already been specified and used for this function, and is in conflict with the pragma environment directive. The latter is ignored.

User Response: Remove the pragma linkage or environment directive.

CCN3724 Undefined identifier ″&1″ was referenced in #pragma &2 directive.

Explanation: A #pragma is referring to an identifier that has not been defined.

User Response: Define the identifier or remove the #pragma.

CCN3728 Operation between types ″&1″ and ″&2″ is not recommended.

Explanation: The operation specified is improper between the operands having the given types. (Accepted.)

User Response: Either change the operator or the operands.

CCN3729 ″&1″ must not be declared inline or static.

Explanation: Although ″&1″ is not a keyword, it is a special function that cannot be inlined or declared as static.

User Response: Remove the inline or static specifier from the declaration of ″&1″.

CCN3730 The pragma is accepted by the compiler. The pragma will have no effect.

Explanation: The pragma is not supported by this compiler.

User Response: The pragma can be removed if desired.

CCN3731 The &1 keyword is not supported on the target platform. The keyword is ignored.

Explanation: The specified keyword is not supported on the target platform and will have no effect.

CCN3732 #pragma &1 is not supported on the target platform.

Explanation: The specified #pragma is not supported on the target platform and will have no effect. See the C/C++ Language Reference for the list of valid #pragma directives.

User Response: Change or remove the #pragma directive.

CCN3733 Processing #include file &1.

Explanation: This message traces #include file processing.

User Response: No response required.

CCN3735 Suboption &1 of &2 ignored because &3 is specified.

Explanation: Suboption &1 of &2 cannot be specified with option &3. &1 is ignored.

User Response: Remove the suboption &1 or the option &3.

CCN3736 &1 conflicts with previous &2 declaration.

Explanation: The compiler cannot resolve the conflicting declarations.

User Response: Remove one of the declarations.

CCN3737 The preprocessor macro ″&1″ was expanded inside a pragma directive.

Explanation: A macro was expanded in the context of a pragma directive. Please ensure that this is the desired result.

User Response: Ensure that the macro was intended for expansion.

CCN3739 Cannot create/use precompiled header file because of memory address space conflict. GENPCH/USEPCH options are ignored.

Explanation: (1) If this a USEPCH compile, the PCH address space (heap area) is not the same as in the GENPCH compile. (2) If this is a GENPCH compile, the persistent heap area is full. In either case, the compilation will continue by ignoring the GENP/USEP options.

User Response: (1) If this is a USEP compile, make sure all the options/pragmas are the same as in GENPCH compile, and the run time environment of the compiler is the same (e.g. region size). (2) If this is a GENP compile, try to reduce the number/size of #include files in the initial sequence.

CCN3740 Timestamp information is not available for #include header file. &1

Explanation: Timestamp information must be present in ALL #include header files when using PCH. Timestamp is absent in sequential datasets, and maybe absent PDS.

User Response: Change any sequential dataset header files into a PDS member. Make sure all PDS member header files contain timestamp information.

CCN3741 Cannot use precompiled header file because #pragmas mismatch before the Initial Sequence.

Explanation: #pragmas appearing before the Initial Sequence must be the same between the GENP and USEP compile.

User Response: Make sure the #pragmas before the Initial Sequence are the same. Use GENPCH to regenerate the PCH file would also solve the problem.

CCN3750 Value of enumeration constant must be in range of signed long.

Explanation: If an enum constant is initialized in the definition of an enum tag, the initial value must be an integral expression that has a value representable as an long.

User Response: Remove the initial value, or ensure that it is an integral constant expression that has a value representable as an long.

CCN3751 Value of enumeration constant must be in range of unsigned long.

Explanation: If an enum constant is initialized in the definition of an enum tag, the value that it is initialized to must be an integral expression that has a value representable as an long.

User Response: Remove the initial value, or ensure that it is an integral constant expression that has a value representable as an long.

CCN3752 Number of enumerator constants exceeds &1.

Explanation: The number of enumerator constant must not exceed the value of &1.

User Response: Remove additional enum constants.

CCN3754 The parameter type is not valid for a function of this linkage type

Explanation: The linkage type of the function puts certain restrictions on the parameter type, on which the function definition violated.

User Response: Check the linkage type restrictions and change the parameter type.

CCN3755 The &1 option is not supported in this release.

Explanation: The specified option is not supported in this release.

User Response: Remove the option.

CCN3763 Option &1 ignored because #pragma &2 is specified.

CCN3764 Option &1 ignored for variable &2 because #pragma &3 is specified.

CCN3765 &1 digits are required for the universal-character-name ″&2″.

CCN3766 The universal-character-name ″&1″ is not in the allowable range for an identifier.

CCN3767 Packed decimal constant &1 is not valid.

Explanation: See the C/C++ Language Reference for a description of a packed decimal constant.

User Response: Ensure that the packed decimal constant does not contain any characters that are not valid.

CCN3769 Static initialization of the 8-byte pointer is not allowed.

Explanation: Static initialization of an 8-byte pointer is only allowed when the teraspace storage model is used.

User Response: Change the STGMDL option to *TERASPACE or use a 16-byte pointer.

CCN3770 The ″__ptr64″ qualifier is not supported in a non-teraspace-enabled environment.

Explanation: The ″__ptr64″ qualifier is only allowed when teraspace is enabled.

User Response: Change the TERASPACE option to *YES or remove the ″__ptr64″ qualifier.

CCN3771 #pragma datamodel(LLP64) directive is not supported in a non-teraspace-enabled environment.

Explanation: #pragma datamodel(LLP64) directive is only allowed when teraspace is enabled.

User Response: Change the TERASPACE option to *YES or remove the #pragma datamodel(LLP64) directive.

CCN3772 An illegal typedef was specified for an AS/400 pointer type.

Explanation: The typedef named for AS/400 pointer types must be 16-byte void pointers.

User Response: Use typedefs of 16-byte void pointers for AS4/00 pointer types.

CCN3773 The __ptr64 qualifier cannot be used with a AS/400 pointer type.

Explanation: AS400 pointer types must be 16-byte in size.

User Response: Remove the __ptr64 qualifier.

CCN3775 The #pragma datamodel directive must appear at file scope.

Explanation: #pragma datamodel must be specified at file scope.

User Response: Move the directive so that it appears at file scope.

CCN3776 The required conditions for using the builtin function ″&1″ are not met.

Explanation: The builtin function ″&1″ requires one or more compiler options that are not currently active.

User Response: Specify the correct options to use the builtin function.

CCN3777 The parameter in position &1 must be a constant literal for the builtin function ″&2″.

Explanation: The builtin function ″&2″ requires parameter &1 to be a constant literal.

User Response: Specify a constant literal for the parameter.

CCN3778 Type ″&1″ is not valid. Type specifier ″&2″ is assumed.

Explanation: The type ″&1″ is not valid; it is treated as ″&2″.

User Response: Replace the unknown type specifier with a correct one.

CCN3790 A threadprivate directive must appear at file scope.

Explanation: #pragma omp threadprivate must be specified at file scope.

User Response: Move the directive to the file scope.

CCN3791 An ordered directive is only allowed within the dynamic extent of for or parallel for that has an ordered clause.

Explanation: An ordered construct can appear only within construct that has ordered clause specified.

User Response: Specify ordered clause on the enclosing for or parallel for.

CCN3792 #pragma &1 may affect behavior of nested or enclosing OpenMP constructs.

Explanation: Pragma may be in conflict with OpenMP functionality.

User Response: Remove pragma it is enclosing or is nested within OpenMP construct.

CCN3793 Option &1 may cause behavior that is different from the one described in OpenMP API Specification.

Explanation: Option may be in conflict with OpenMP.

User Response: Remove the option.

CCN3794 Atomic directive is not followed by an expression statement of the required form.

Explanation: Atomic directive has to be followed by a compound assignment expression or increment/decrement expression statement.

User Response: Correct the statement.

CCN3795 Private variable ’&1’ appears in the &2 clause.

Explanation: Private variable cannot appear in that clause.

User Response: Remove variable from the clause.

CCN3796 &1 construct cannot be nested within &2 construct.

Explanation: OpenMP constructs are incorrectly nested.

User Response: Correct the constructs.

CCN3797 &1 directive cannot appear within &2 construct.

Explanation: Directive is incorrectly nested.

User Response: Correct the directive.

CCN3798 Threadprivate variable ’&1’ appears in the &2 clause.

Explanation: Threadprivate variable cannot appear in that clause.

User Response: Remove variable from the clause.

CCN3799 OpenMP constructs cannot be used with optimization level 0. Optimization level 2 will be assumed.

Explanation: Optimization level 0 was specified for a function that contains OpenMP construct.

User Response: Change the optimization level or remove the construct.

CCN3805 String literal exceeded the compiler limit of &1.

Explanation: String literal size cannot be larger than the compiler limit

User Response: Reduce the size of the string literal.

CCN3810 #pragma runopts syntax (&1): &2

Explanation: Syntax error in the pragma. The suboption syntax is the same as the corresponding LE runtime option. Please refer to the LE manual for details of the CEEnnnn message number.

User Response: Correct the syntax error.

CCN3811 Option &1 forces &2 to take effect.

Explanation: The first option in the message forces the second one to take effect. Specify the second option explicitly to suppress this message.

User Response: Specifiy the second option explicitly.

CCN3812 Option FLOAT(IEEE) may cause slow execution time when used with ARCH less than 3.

Explanation: Binary floating point operations (BFP) needs hardware architecture (ARCH option) of 3 or higher. For ARCH less than 3, BFP will work on OS level V2R6 or higher, which provides software emulation, but will significantly slow down the execution time.

User Response: If the target hardware architecture is 3 or higher, specify it explicitly in ARCH.

CCN3813 Option FLOAT(AFP) may cause slow execution time when used with ARCH less than 3.

Explanation: The AFP suboption needs hardware architecture (ARCH option) of 3 or higher. For ARCH less than 3, BFP will work on OS level V2R6 or higher, which provides software emulation, but will significantly slow down the execution time.

User Response: If the target hardware architecture is 3 or higher, specify it explicitly in ARCH.

CCN3815 Conflicting qualifiers &1 and &2 specified.

Explanation: The identified qualifiers cannot both be specified at the same time.

User Response: Remove one of the qualifiers.

CCN3862 Unable to read &1.

Where: &1 file *CHAR 100

Explanation: The compiler encountered an error while reading from the specified file.

CCN3863 Unable to write to &1.

Where: &1 file *CHAR 100

User Response: Ensure that the disk drive is not in an error mode and that there is enough disk space left.

CCN3870 The program name &1 has been truncated to &2.

Explanation: The program name exceeds the maximum length of 10 characters and has been truncated. This may result in unexpected behavior if two different names become the same name after truncation.

User Response: Reduce the length of the program name. Alternatively, use #pragma map to shorten program name.

CCN3871 #pragma &1 was ignored for function or typedef &2.

User Response: Verify what has been specified and remove the conflicts.

CCN3885 An anonymous union or struct declared at file scope must be static.

Explanation: Anonymous tag at file scope level cannot be non-static.

User Response: Declare all anonymous tags to be static in file scope level.

CCN3886 Member ″&1″ is at offset ″&2″, not at offset ″&3″ as specified in #pragma assert_field_offset.

Explanation: The offset of member &1 is not at the offset specified by the pragma.

User Response: Either fixing the aggregate which contains the member or fixing the offset in the pragma.

CCN3887 The first operand in #pragma assert_field_offset must be a struct, a union, or a typedef of a struct or a union. The pragma is ignored.

Explanation: The #pragma assert_field_offset cannot be anything other than a struct, a union, or a typedef of a struct or a union.

User Response: Change the first operand or remove the pragma.

CCN3888 The first operand of #pragma assert_field_offset is incomplete. The pragma is ignored.

Explanation: An incomplete struct or union is not a valid argument to ″#pragma assert_field_offset″. A structure or union type is completed when the definition of its tag is specified.

User Response: Ensure the struct or union is a complete type.

CCN3889 Member ″&1″ is not declared as specified in #pragma ″assert_field_offset″. The pragma is ignored.

Explanation: The specified member does not belong to the structure or union specified in the pragma.

User Response: Make sure the member is in the structure or union specified in the pragma.

CCN3890 The declaration ″&1″ specified in ″#pragma assert_field_offset″ cannot be found. The pragma is ignored.

Explanation: The declaration ″&1″ specified in ″#pragma assert_field_offset″ has not been declared.

User Response: Declare the type.

CCN3891 Subscript operator requires an array operand in ″#pragma assert_field_offset″. The pragma is ignored.

Explanation: A subscript was specified in ″#pragma assert_field_offset″ but the operand is not an array.

User Response: Either change the operand to be an array type or remove the subscript operator.

CCN3892 Array index must be a constant expression in ″#pragma assert_field_offset″. The pragma is ignored.

Explanation: The ″#pragma assert_field_offset″ is evaluated at compile time. Thus all arguments must be constant expressions.

User Response: Change the expression.

CCN3894 The &1 is not valid in 64-bit mode and it is ignored.

Explanation: The &1 is not valid in 64-bit mode. It is only supported in 32-bit mode.

User Response: Either remove &1 or compile it in 32-bit mode.

CCN3895 An aggregate containing long double on IA64 platform might not have the same size/alignment compared to that on IA32 platform.

Explanation: An aggregate containing long double on IA64 platform might not have the same size/alignment compared to that on IA32 platform.

User Response: Replacing long double in the aggregate or change to ia64 alignment rule.

CCN3896 Operand has type &1.

Where: &1 C type *CHAR 100

Explanation: An error has occurred due to conflicting operands. This message states the type of the operand used in the expression.

User Response: No recovery is necessary if this result was intended. Change the type of the operand if necessary.

CCN3897 Unstructured goto statement encountered.

Explanation: The target label of a goto statement should not be located in an inner block such as a loop.

User Response: Ensure the target label of the goto statement is not located in an inner block.

CCN3898 The #include header &1 is not valid.

Where: &1 header *CHAR 100

Explanation: The name of the file specified on the #include directive is not valid.

User Response: Remove or correct the #include directive.

CCN3913 The enum constants must be specified when the enum tag is declared.

Explanation: When an enumeration tag is declared, the list of the enumeration constants must be included in the declaration.

User Response: Add the list of enumeration constants in the enum tag declaration.

CCN3919 Variable &1 was not explicitly initialized.

Explanation: If not explicitly initialized, variables with storage class auto or register contain indeterminate values.

User Response: Initialize the variable.

CCN3920 Bitwise operator applied to a signed type.

Explanation: Bitwise operators may change the value of a signed type by shifting the bit used to indicate the sign of the value.

User Response: Change the operand to an unsigned type or remove the bitwise operation.

CCN3929 The usage of type generic macro ″&1″ conflicts with the previous declaration of ″&2″.

Explanation: There is a prototype or defintion of function ″&1″ which differs from the prototype of the ISO C9X type generic macro.

User Response: Remove the prototype or defintion.

CCN3930 The function ″&1″ is not a type generic macro.

Explanation: One should not use the __tg_builtin for non- type generic macro.

User Response: Remove the call to __tg_builtin.

CCN3931 Dependency file &1 cannot be opened.

Where: &1 is a file name.

Explanation: Makedepend could not open the specified dependency file.

User Response: Ensure the source file name is correct. Ensure that the correct file is being read and has not been corrupted. If the file is located on a LAN drive, ensure the LAN is working properly. Also, the file may be locked by another process or access may be denied because of insufficient permission

CCN3932 Too few option specified for makedepend.

User Response: Specify correct number of options for makedepend.

CCN3933 Specify at least one source operand to be processed.

CCN3934 Compiler option &1 is invalid for compiler version &2.

Where: &1 is a compiler option, &2 is compiler version.

Explanation: An invalid option was specified for the compiler version specified for makedepend.

User Response: Change the compiler version to the version that accepts this option, or remove this option.

CCN3935 Specify a valid -W phase code (0 or c=compile, m=makedepend) instead of &1.

Explanation: An invalid compiler phase was specified for makedepend.

User Response: Specify a phase that is accepted by makedepend.

CCN3936 Specify a series of options, separated by commas, for the -W m option.

CCN3937 &1 has a dependency on include file &2 which is located in an MVS data set.

Where: &1 is an object file. &2 is a #include file.

Explanation: The specified #include file was found in an MVS data set. No dependency information will be recorded for this #include file.

CCN3938 Unknown compiler version &1 for makedepend option V. Using default compiler version.

Where: &1 is a compiler version.

Explanation: An invalid compiler version was specified for the makedepend option V.

User Response: Correct the compiler version.

CCN7500 The option ″%1$s″ is not supported.

Where: %1$s is an option.

Explanation: The command line contained an option that is not supported. Note that some option parameters must not have spaces between the option and the parameter.

User Response: Remove the option. Check the syntax of the options.

CCN7501 Suboption ″%1$s″ for option ″%2$s″ is not supported on the target platform.

Where: %1$s is the suboption. %2$s is the option.

Explanation: The option has been specified with a suboption that is not supported on the target platform.

User Response: change the suboption, or remove the option.

CCN7502 Missing value for option ″%1$s″.

Where: %1$s is an option name

Explanation: The option was missing a required parameter. See the ″User's Guide″ for details on the option.

User Response: Add a value for the option.

CCN7503 Unrecognized value ″%1$s″ specified with option ″%2$s″.

Where: %1$s is the value specified with the option,. %2$s is the option name.

Explanation: An inappropriate value was used with the option.

User Response: Remove the unrecognized value.

CCN7504 ″%1$s″ is not a valid suboption for ″%2$s″. The option is ignored.

Where: %1$s is the suboption, %2$s is the option.

Explanation: The command line contained an option with an invalid suboption.

User Response: Remove the suboption.

CCN7505 The value given for the ″priority″ option is in the range reserved for the system.

Explanation: Priority values less than -2147482624 are reserved for system purposes.

User Response: Change the priority value so that it is greater than -2147482624.

CCN7506 ″%1$s″ is no longer supported. The option is ignored.

Where: %1$s is the outdated option.

Explanation: The command line contained an option that is no longer supported by this release.

User Response: Remove the option.

CCN7507 Options ″%1$s″ and ″%2$s″ are not compatible.

Where: %1$s and %2$s are both option names.

Explanation: The specified options cannot be used together.

User Response: Change option values.

CCN7508 Suboption ″%1$s″ for option ″%2$s″ is no longer supported. The suboption is ignored.

Where: %1$s is the suboption. %2$s is the option.

Explanation: The command line contained a suboption that is no longer supported by this release.

User Response: Remove the suboption.

CCN7509 The suboption specified for the ″%1$s″ option is not allowed when the ″%2$s″ option is specified.

Where: %1$s and %2$s are option names.

Explanation: The suboption specified in the first option conflicts with the second option. The first option is ignored.

User Response: Correct the conflicting option or suboption.

CCN7510 Insufficient memory.

Explanation: The available memory has been exhausted.

User Response: Provide more memory.

CCN7511 Either the default or user-defined maximum number of error messages has been exceeded.

Explanation: There have been too many errors to continue.

User Response: Fix the previous errors.

CCN7512 Compiler cannot create temporary files. The file system may be full or not writable.

Explanation: The intermediate code files could not be created. Please verify that the target file system exists, is writable, and is not full.

User Response: Ensure that the designated location for temporary objects exists, is writable, and is not full.

CCN7513 An error was detected while writing to an temporary file. The file system may be full.

Explanation: An error occurred writing to an intermediate code file. Please verify that the target file system exists, is writable, and is not full.

User Response: Ensure that the designated location for temporary objects exists, is writable, and is not full.

CCN7517 The template registry file ″%1$s″ could not be opened.

Where: ″%1$s″ is the template registry file name designated by the templateregistry compiler option.

Explanation: A template registry file is created when the templateregistry compiler option is enabled.

User Response: Ensure that file system permissions allow files to be written, and that sufficient file system resources exist to permit the creation of this file.

CCN7518 Error reading template registry file ″%1$s″.

Where: ″%1$s″ is the template registry file name designated by the templateregistry compiler option

Explanation: The template registry file is corrupt.

User Response: Delete the template registry file and recompile all of the source files using this registry.

CCN7519 Error writing to template registry file ″%1$s″.

Where: ″%1$s″ is the template registry file name designated by the templateregistry compiler option.

Explanation: A template registry file is created when the templateregistry compiler option is enabled.

User Response: Ensure that file system permissions allow files to be written, and that sufficient file system resources exist to permit the creation of this file.

CCN7599 The compiler could not open the output file ″%1$s″.

Where: %1$s is a file name.

User Response: Ensure the output file name is correct. Also, ensure that the location of the output file has sufficient storage available. If using a LAN drive, ensure that the LAN is working properly and you have permission to write to the disk.

CCN7601 Goto statements should not be used.

Explanation: Goto statements are often lead to difficult to maintain code.

User Response: Remove the goto statements

CCN7602 Ellipsis notation should not be used.

Explanation: Using ellipsis prevents type checking of arguments.

User Response: Remove ellipsis

CCN7607 ″%1$s″ should probably define a constructor.

CCN7608 ″%1$s″ should probably define a destructor.

CCN7609 ″%1$s″ should probably define a copy constructor.

CCN7611 Argument ″%1$s″ is not used in function ″%2$s″.

Where: ″%1$s″ is used argument and ″%2$s″ where the argument is declared

Explanation: The argument ″%1$s″ is specified but not needed

User Response: Consider removing the argument from the paramater list of the function

CCN7612 ″%1$s″ is set but not used in function ″%2$s″.

Where: ″%1$s″ is the variable that is set but not used and ″%2$s″ is the function where the variable resides.

Explanation: A variable has been explicitly initialized or assigned but is not referenced

User Response: Remove the variable if there are no side-effects.

CCN7613 The destructor in the base class of ″%1$s″ should be made virtual.

Where: ″%1$s″ is the base class to change.

Explanation: A virtual destructor in the base class ensures that the proper destructor is called.

User Response: Declare the destructor with the virtual keyword.

CCN7614 A user-defined copy constructor/assignment operator should be created in ″%1$s″ to handle a pointer data member.

Where: ″%1$s″ is the class that has a pointer to data member

Explanation: The compiler generated copy constructor and assignment operator does a bitwise member copy.

User Response: Create a copy constructor and an assignment operator

CCN7616 ″%1$s″ does not assign values to all data members in the class.

Where: ″%1$s″ is the offending class

Explanation: Checks that all data members in a class are assigned to when user defined assignment operators are present

User Response: Assign value to data member

CCN7617 ″%1$s″ was not assigned.

Explanation: Data member that is not assigned

User Response: Assign value to data member

CCN7618 ″%1$s″ should be initialized using the member initialization list.

Where: ″%1$s″ is the data member to initialize

Explanation: Initializing a data member is faster than assignment in the constructor

User Response: Initialize data member in the constructor list

CCN7619 ″%1$s″ should be initialized in the same order as it is declared in ″%2$s″.

Where: ″%1$s″ is the data member to re-order

Explanation: Data members are initialized in the order they are declared, the initialization list should reflect this.

User Response: Re-order the initialization list to make the declaration order

CCN7620 ″%1$s″ is a non-const namespace variable and may cause problems in multi-threaded code.

Where: ″%1$s″ is a variable in namescope scope

Explanation: Variables in namespace scope that are not protected by a mutex may behave unexpectedly in multi-threaded code

User Response: Don't use variables in namespace scope for multi-threaded code

CCN7621 ″%1$s″ is a global variable and may cause problems in multi-threaded code.

Where: ″%1$s″ is a global variable

Explanation: Global variables that are not protected by a mutex may behave unexpectedly in multi-threaded code

User Response: Don't use global variables for multi-threaded code

CCN7622 ″%1$s″ is a static local variable and may cause problems in multi-threaded code.

Where: ″%1$s″ is a static local variable

Explanation: Static local variables that are not protected by a mutex may behave unexpectedly in multi-threaded code

User Response: Don't use static local variables for multi-threaded code

CCN7623 ″%1$s″ is a static member variable and may cause problems in multi-threaded code.

Where: ″%1$s″ is a static member variable

Explanation: Static member variables that are not protected by a mutex may behave unexpectedly in multi-threaded code

User Response: Don't use static member variables for multi-threaded code

CCN7624 64-bit portability : possible truncation of pointer through conversion of pointer type into int type

Explanation: Possible truncation of pointer through conversion of pointer type into int type

User Response: Change the int type to long

CCN7625 64-bit portability : possible truncation of array through conversion of array type into int type

CCN7626 64-bit portability : possible truncation of function through conversion of function type into int type

CCN7627 64-bit portability : possible incorrect pointer through conversion of int type into pointer

CCN7628 64-bit portability : possible loss of digits through conversion of long type into int type

CCN7629 64-bit portability : possible difference in results. In 32-bit mode values greater than INT_MAX would be truncated, but not in 64-bit mode.

CCN7630 64-bit portability : possible difference in results. In 32-bit mode values < INT_MIN or > INT_MAX would be truncated, but not in 64-bit mode.

CCN7631 64-bit portability : possible difference in results. Values < 0 would give different results in 64-bit mode, values > UINT_MAX would be truncated in 32-bit mode but not in 64-bit mode.

CCN7632 64-bit portability : possible difference in results. Values > INT_MAX would be truncated in 32-bit mode but not in 64-bit mode.

CCN7633 64-bit portability : possible difference in results. Values > UINT_MAX would be truncated in 32-bit mode but not in 64-bit mode.

CCN7634 64-bit portability : possible difference in results if value is negative.

CCN7635 ″%1$s″ is not used in function ″%2$s″.

CCN7636 Global variable ″%1$s″ is not used.

Where: The global variable.

Explanation: A global variable was declared but not used.

User Response: Remove the variable.

CCN7637 Null statement.

Explanation: This C++ statement has no effect.

User Response: Remove the statement.

CCN7638 The condition evaluates to a constant value.

Explanation: An expression in a condition will not change during execution.

User Response: Remove the condition.

CCN7639 Precision will be lost in assignment to bit-field ″%1$s″

Where: The name of the bit-field.

Explanation: The size of the value assigned to the bit-field is too large.

User Response: Increase the size of the bit-field or reduce the value assigned.

CCN7640 The statement is unreachable

Explanation: Statements that are unreachable are never executed.

User Response: Remove unreachable statements.

CCN7641 Auto compiler temporary of type ″%1$s″ has been generated.

Where: The type of the temporary variable.

Explanation: A temporary variable was generated by the compiler to hold an intermediate result.

User Response: Modify expression to remove the need for the compiler generated temporary.

CCN5001 A typedef must not have an initializer.

Explanation: A typedef represents a type, and a type must not have an initializer.

User Response: Remove the initializer.

CCN5002 A typedef must not be specified on a function.

Explanation: A typedef represents a type and must not be specified on a function definition.

User Response: Remove the typedef keyword.

CCN5003 A destructor must be a class member.

Explanation: A destructor is a special member function that cannot be declared outside a class declaration.

User Response: Remove the destructor declaration or move it inside the class declaration.

CCN5004 A conversion operator must be a class member.

Explanation: A conversion operator is a special member function that converts an object of the class type to an object of the conversion type.

User Response: Move the conversion operator declaration inside the class from which you want to convert.

CCN5005 ″%1$s″ must have ″C″ linkage.

Where: ″%1$s″ is the string representing the main function.

Explanation: The main function ″%1$s″ cannot be specified with any linkage type other than extern ″C″.

User Response: Remove the linkage specification or change it to extern ″C″.

CCN5006 The function declaration must not have a body.

Explanation: A declaration of a function in a lexical scope can only declare a function and must not be a definition.

User Response: Remove the function body from the declaration.

CCN5007 A bit-field is not allowed here.

Explanation: Bit-fields can only be declared as members of classes.

User Response: Move the declaration into a member list.

CCN5008 An initializer is not allowed here.

Explanation: A function declaration cannot have an initializer.

User Response: Remove the initializer.

CCN5009 A union must not have base classes.

Explanation: Only a struct or a class can have a base class.

User Response: Change the union to a class or struct.

CCN5010 A name must not be used more than once within a template parameter list.

Explanation: Duplicated template parameter names are not allowed.

User Response: Change the name of one of the template parameters.

CCN5011 ″%1$s″ is not a namespace.

Where: ″%1$s″ is the name used in the source.

Explanation: Only namespaces can be used in using directives, but the entity named is not a namespace.

User Response: Remove the using directive or change the name to be that of a namespace.

CCN5012 A using declaration for a member is allowed only in a class or struct.

Explanation: The using declaration is in a union, but using declarations are only allowed in classes and structs.

User Response: Remove the using declaration.

CCN5013 ″%1$s″ is not a destructor.

Where: ″%1$s″ is the name in error.

Explanation: The name following the ″˜″ must denote a destructor when it is used in a member list, but the name specified is not a destructor.

User Response: Change the name to be a destructor.

CCN5014 The literal type is unknown.

Explanation: The type of literal specified is not recognised.

User Response: Change the literal to a recognised type.

CCN5015 A constant expression is expected.

Explanation: A constant expression can only be used in this context.

User Response: Change the expression to be a constant expression.

CCN5016 The expression must be an integral constant expression.

Explanation: Only a constant expression can be used in this context, but a non-constant expression is specified.

User Response: Change the expression to be a constant expression.

CCN5017 A class or struct declaration must have a class name, a declarator, or both.

Explanation: Anonymous classes and structs are extensions to the language and may result in code that is not portable to other compilers.

User Response: Name the class or add a declarator list.

CCN5018 An enumeration must not be a template.

Explanation: A template can only be a class, struct, or function.

User Response: Remove the template keyword and template arguments, or nest the enumerator within a template.

CCN5019 A typedef declaration must not be a template.

Explanation: A template can only be a class, struct, or function.

User Response: Remove the template keyword and template arguments, or nest the typedef within a template.

CCN5020 A bit-field must not have a ″%1$s″ specifier.

Where: ″%1$s″ is the specifier that is not valid for a bit-field.

Explanation: A bit-field should have integral or enumeration type, and it should not be static.

User Response: Remove the incorrect specifier from the bit-field or use an array rather than a bit-field.

CCN5021 The named class is not defined.

Explanation: The class named in the elaboration is qualified but does not exist.

User Response: Change the name to refer to a declared class.

CCN5022 The named class is not a class name.

Explanation: The name specified in the elaboration is not a class or struct.

User Response: Change the name to be a class or struct, or remove the elaboration.

CCN5023 The named struct is not defined.

Explanation: The struct named in the elaboration is qualified but does not exist.

User Response: Change the name to refer to a declared struct.

CCN5024 The named struct is not a struct name.

Explanation: The name specified in the elaboration is not a class or struct.

User Response: Change the name to be a class or struct.

CCN5025 The named union is not defined.

Explanation: The union named in the elaboration is qualified but does not exist.

User Response: Change the name to refer to a declared union.

CCN5026 The named union is not a union name.

Explanation: The name specified in the elaboration is not a union.

User Response: Change the name to be a union.

CCN5027 A function template must not be a qualifier.

Explanation: Qualifiers can only be namespaces or classes.

User Response: Correct the qualifier name or remove it.

CCN5028 A qualified name is not allowed in the definition of ″%1$s″.

Where: ″%1$s″ is the name in error.

Explanation: A name specified as a parameter, in a enumeration definition, or as an enumerator must not be a qualified name.

User Response: Remove the qualifiers from the name.

CCN5029 The named enumeration is not defined.

Explanation: Either the enumeration named in the elaboration is not defined or a forward declaration of an incorrect enumeration is being attempted.

User Response: Change the name to be a defined enumeration or define the enumeration.

CCN5030 The named enumeration is not an enumeration name.

Explanation: The name specified in the elaboration is not an enumeration.

User Response: Change the name to be an enumeration.

CCN5031 A function template must not be the class referred to by a pointer-to-member.

Explanation: Only classes can form pointer-to-members.

User Response: Correct the class or remove the pointer-to-member.

CCN5032 The destructor name is not valid.

Explanation: A destructor name cannot be a qualifier.

User Response: Change the name to be a destructor.

CCN5033 A typedef declaration must declare a name.

Explanation: A typedef declaration declares a type but no name is specified for the declaration.

User Response: Add a name to the typedef declaration.

CCN5034 This message is no longer used.

CCN5035 A simple namespace name is expected.

Explanation: The name specified in a namespace declaration or a namespace alias cannot be qualified.

User Response: Remove the qualifiers from the name.

CCN5036 A namespace name is expected.

Explanation: The name specified in the namespace alias declaration must refer to a namespace.

User Response: Change the name to be a namespace name.

CCN5037 A qualified name is expected in a using declaration.

Explanation: An unqualified name has been specified in a using declaration. A using declaration must nominate a member of a namespace or class.

User Response: Change the name to be a qualified name.

CCN5038 The name ″%1$s″ is not a type.

Where: ″%1$s″ is the name in error.

Explanation: The name is elaborated with ″typename″ but the name specified in the template instantiation is not a type.

User Response: Change the name to refer to a type in the instantiation.

CCN5039 A label must be a simple identifier.

Explanation: The label specified was a qualified name, but only unqualified names can be used for labels.

User Response: Remove the qualifiers from the label.

CCN5040 The text ″%1$s″ is unexpected. ″%2$s″ may be undeclared or ambiguous.

Where: ″%1$s″ is the symbol causing the syntax error. ″%2$s″ is the name that may be causing the error if it is expected to be a type.

Explanation: There is a syntax error in the declaration. It may be that a name that is expected to be a type is unknown or ambiguous.

User Response: Remove the offending symbol or ensure that the name used as a type name is actually a type.

CCN5041 A pointer-to-member must not be specified because ″%1$s″ is not a class.

Where: ″%1$s″ is the erroneous class type.

Explanation: The final qualifier in a pointer-to-member must be a class.

User Response: Change the final qualifier to be a class.

CCN5042 ″auto″ must be used only in a lexical block or for parameters.

Explanation: The keyword ″auto″ must be used only in a function body or for function parameters.

User Response: Remove the ″auto″ specifier.

CCN5043 ″register″ must be used only in a lexical block or for parameters.

Explanation: The keyword ″register″ must be used only in a function body or for function parameters.

User Response: Remove the ″register″ specifier.

CCN5044 Only function declarations can have default arguments.

Explanation: A default initializer has been specified in the parameter list of a function but the function is not being declared.

User Response: Remove the default initializers.

CCN5045 The type of a conversion constructor must not contain the specifier ″%1$s″.

Where: ″%1$s″ is the invalid specifier.

Explanation: The first parameter of a conversion constructor must not use the specifier ″%1$s″.

User Response: Remove the specifier.

CCN5046 The attributes ″%1$s″ must not be specified for a parameter.

Where: ″%1$s″ is the invalid specifier.

Explanation: It is not valid to specify the attribute ″%1$s″ for a function paarameter or template parameter.

User Response: Remove the specifier.

CCN5047 A template class declaration or definition must have a class name.

Explanation: Anonymous class templates are not allowed.

User Response: Add a name.

CCN5048 A name is expected in the template parameter.

Explanation: The template parameter must have a name.

User Response: Add a name for the template parameter.

CCN5049 A template function must not be explicitly specialized as a class.

Explanation: A template function can only be specialized as a function.

User Response: Correct the specialization or the template.

CCN5050 The syntax is not valid.

Explanation: The compiler does not recognize the syntax used. There may be a typing error.

User Response: Fix the syntax.

CCN5051 A template parameter must be a simple identifier.

Explanation: A template parameter is a type parameter or a parameter declaration.

User Response: Correct the template parameter name.

CCN5052 ″typedef″ must not appear with a storage class specifier.

Explanation: It is not valid to have specifiers in the auto, register, static, extern, or mutable declaration of a typedef.

User Response: Remove the typedef or the storage class specifier.

CCN5053 The declaration of a class member within the class definition must not be qualified.

Explanation: A class member that is declared in the member list of a class must not be a qualified name.

User Response: Remove the qualifier.

CCN5054 A class or struct declaration must have a tag, a declarator, or both.

Explanation: Anonymous classes and structs are extensions to the language, and the option allowing them is turned off.

User Response: Name the class, add a declarator list, or use the appropriate language level option to allow anonymous structs.

CCN5055 ″%1$s″ is specified more than once.

Where: ″%1$s″ is the extra specifier.

Explanation: The specifier ″%1$s″ is used in the declaration more than once but the extra specifiers are ignored.

User Response: Remove the extra specifiers.

CCN5056 A constructor must not be a pure virtual function.

Explanation: Constructors cannot be virtual, but the constructor has been specified as a pure virtual function.

User Response: Remove the ″=0″ from the constructor declaration.

CCN5057 The declaration specifier is missing.

Explanation: Implicit int types are no longer valid in C++.

User Response: Add a complete type to the declaration or use the appropriate language level option to allow implicit int types.

CCN5058 The declaration of a class member within the class definition must not be qualified.

Explanation: A class member that is declared in the member list of a class must not be a qualified name.

User Response: Remove the qualifier.

CCN5059 The identifier ″%1$s″ appears where a template identifier is expected.

Where: ″%1$s″ is the unexpected identifier.

Explanation: The compiler expects a template identifier, and ″%1$s″ is not a template.

User Response: Correct the identifier or its use.

CCN5060 An internal parser error has occurred: ″%1$s″.

Where: ″%1$s″ is a description of the error.

Explanation: The parser has detected an unrecoverable error.

User Response: Report the problem to your IBM C++ service representative.

CCN5061 This message is no longer used.

CCN5062 The incomplete class ″%1$s″ must not be used as a qualifier.

Where: ″%1$s″ is the incomplete class.

Explanation: A class that is incomplete because it is only declared or because of some error in the declaration cannot be used as a qualifier.

User Response: Define the class.

CCN5063 The text ″%1$s″ is unexpected.

Where: ″%1$s″ is the first invalid token.

Explanation: A syntax error has occurred and the first unexpected token is ″%1$s″.

User Response: Change or remove the offending text.

CCN5064 Syntax error: ″%1$s″ was expected but ″%2$s″ was found.

Where: ″%2$s″ is the invalid text. ″%1$s″ is expected correct text.

Explanation: A syntax error has occurred and the first unexpected token is ″%1$s″. The only valid token at this point is ″%2$s″.

User Response: Change the incorrect token to the expected one.

CCN5065 The qualifier ″%1$s″ is neither a class nor a namespace.

Where: ″%1$s″ is the invalid qualifier.

Explanation: Only names representing classes and namespaces can be used as qualifiers.

User Response: Change the qualifier to a class name or namespace name.

CCN5066 A function must not be defined in this scope.

Explanation: Function definitions are only allowed in namespace scope or in a member list of a class.

User Response: Move the definition into an appropriate scope.

CCN5067 A return type must not be specified for ″%1$s″.

Where: ″%1$s″ is the function that cannot have a return type.

Explanation: Return types cannot be specified for conversion functions.

User Response: Remove the return type.

CCN5068 No member except a constructor can have the same name as its class, struct, or union.

Explanation: An attempt was made to declare a member of a class that has the same name as the class itself.

User Response: Change the name of the member. ″constructor″ is an ISO-defined term.

CCN5069 The bit-field length must be greater than, or equal to, zero.

Explanation: A bit-field length must not be a negative number.

User Response: Change the bit-field length to zero or a positive number.

CCN5070 The friend class declaration must use the ″%1$s″ keyword in the friend decalaraton of ″%2$s″.

Where: ″%1$s″ is the expected elaboration. ″%2$s″ is the offending text.

Explanation: The language has changed. Now declarations of friend classes must contain an elaborated type specifier.

User Response: Add the elaboration of class, struct, or union to the declaration.

CCN5071 A class or union must not be defined in this context.

Explanation: An attempt was made to define a class in a context where this is not valid.

User Response: Move the definition to an appropriate context.

CCN5073 A template specialization must not be declared here.

Explanation: An explicit specialization can only be declared in namespace scope, either in the namespace in which the primary template is declared or, for a member template, in the namespace of which the enclosing class is declared.

User Response: Remove the specialization or move it to a valid location.

CCN5074 The ″%1$s″ specifier must not be specified for a friend.

Where: ″%1$s″ is the invalid specifier.

Explanation: The ″%1$s″ specifier is not correct on a friend declaration.

User Response: Remove the invalid specifier.

CCN5075 A static member function must not be virtual.

Explanation: The virtual specifier must not be used on a member function that is declared static.

User Response: Remove the virtual or static specifier.

CCN5076 The pure-specifier (= 0) is not valid for a static member function.

Explanation: The pure-specifier must not be used on a member function that is declared static.

User Response: Remove the pure-specifier or static specifier.

CCN5077 The array bound is too large.

Explanation: The specified array bound is too large for the system to handle.

User Response: Use a smaller array bound.

CCN5078 A template must not be defined here.

Explanation: A template can only be defined at namespace or class scope.

User Response: Remove the template definition or move it to a valid location.

CCN5079 The bit-field length is too large.

Explanation: The specified bit-field length is larger than the system allows.

User Response: Use a smaller bit-field length.

CCN5080 Template specializations must be prefixed with ″template<>″.

Explanation: Old-style template specializations are accepted but are no longer compliant.

User Response: Add the ″template <>″ syntax.

CCN5081 The declaration ″%1$s″ is not a template declaration.

Where: ″%1$s″ is the incorrect declaration.

Explanation: The compiler expects a template declaration, and ″%1$s″ is not a template.

User Response: Change the declaration to be a template.

CCN5082 This message is no longer used.

CCN5083 An explicit template specialization must not be an untagged class.

Explanation: An identifier is required for this declaration.

User Response: Supply the identifier of the template that is being explicitly specialized.

CCN5084 An explicit template instantiation must not be an untagged class.

Explanation: An identifier is required for this declaration.

User Response: Supply the identifier of the template that is being explicitly instantiated.

CCN5085 This message is no longer used.

CCN5086 The declaration of the template parameters is missing for template ″%1$s″.

Where: ″%1$s″ is the incorrect template declaration.

Explanation: A template must have at least one template parameter.

User Response: Correct the template parameters or remove the invalid template declaration.

CCN5087 The arguments of the template qualifier do not match those of ″%1$s″.

Where: ″%1$s″ is the matching template declaration.

Explanation: The types and order of template arguments must match the original template.

User Response: Correct the arguments in the template qualifier.

CCN5088 An enumeration must not be defined in this context.

Explanation: An attempt is being made to define an enumeration in a context where it is not valid to define an enumeration.

User Response: Move the definition to an appropriate context.

CCN5089 Too many template prefixes are specified for the declaration of ″%1$s″.

Where: ″%1$s″ is the incorrect declaration.

Explanation: The number of template scopes must match the template nesting level of the declaration.

User Response: Remove some of the template scopes.

CCN5090 Not enough template prefixes are specified for the declaration of ″%1$s″.

Where: ″%1$s″ is the incorrect declaration.

Explanation: The number of template scopes must match the template nesting level of the declaration.

User Response: Add the correct number of template scopes.

CCN5091 A function explicit instantiation must specify only ″template instantiation-name″.

Explanation: You cannot provide a definition or use the pure virtual specification on a function explicit instantiation.

User Response: Correct the function explicit instantiation.

CCN5092 An explicit instantiation must instantiate a template function definition.

Explanation: There must be a function body to instantiate.

User Response: Define the template function or remove the explicit instantiation.

CCN5093 A partial specialization of a function is not allowed.

Explanation: Only class templates can be partially specialized.

User Response: Remove the function partial specialization.

CCN5094 The template parameter must not be qualified.

Explanation: A template parameter defines the parameter to be a type in the scope of the template and therefore cannot be qualified.

User Response: Remove all qualifiers.

CCN5095 The friend function declaration ″%1$s″ will cause an error when the enclosing template class is instantiated with arguments that declare a friend function that does not match an existing definition. The function declares only one function because it is not a template but the function type depends on one or more template parameters.

Where: ″%1$s″ is the non-template friend declaration that depends on template parameters.

Explanation: This friend function makes use of one or more of the enclosing template's parameters. Therefore different instantiations of the template will create different friend functions. If a created friend function does not exist, the program will not link.

User Response: Change the friend declaration to a template function (by adding explicit template arguments) or ensure that all instantiations will match an existing function.

CCN5096 No primary class template ″%1$s″ is found for a partial specialization.

Where: ″%1$s″ is the incorrect class template partial specialization.

Explanation: A primary class template must exist for a partial specialization.

User Response: Declare the primary template or remove the partial specialization.

CCN5097 This message is not implemented.

CCN5098 The partial specialization ″%1$s″ must be declared in the same scope as the primary template or in a namespace scope that encloses the primary template.

Where: ″%1$s″ is the incorrect class template partial specialization.

Explanation: The primary template must be visible at the point the partial specialization is made.

User Response: Move the partial specialization into a correct scope.

CCN5099 The explicit specialization ″%1$s″ must be made in the same scope as the primary template.

Where: ″%1$s″ is the incorrect class template explicit specialization.

Explanation: The primary template must be visible at the point the explicit specialization is made.

User Response: Move the explicit specialization into a correct scope.

CCN5100 The class qualifier ″%1$s″ contains a circular reference back to ″%2$s″.

Where: ″%1$s″ and ″%2$s″ are the classes with circular references.

Explanation: The two classes contain references to each other that require each class to be defined before the other.

User Response: Change one of the classes so that it does not require the other class to be defined.

CCN5101 A typedef declaration must not contain the specifier ″%1$s″.

Where: ″%1$s″ is the invalid specifier.

Explanation: A typedef defines another name to use in place of the declared type. The indicated specifier is not valid in this context.

User Response: Remove the specifier.

CCN5102 A declaration with a ″%1$s″ specifier must contain a declarator ID.

Where: ″%1$s″ is the specifier in question.

Explanation: The type for the declaration contains a specifier that requires an object to be declared.

User Response: Remove the specifier or declare an object.

CCN5103 An anonymous union, struct or class declared at namespace scope must be declared static.

Explanation: Data members of an anonymous union, struct, or class declared at namespace scope have internal linkage so they must be declared static.

User Response: Add the static specifier to the union, struct, or class.

CCN5104 The ″%1$s″ specifier must be applied only to objects declared in a block or to function parameters.

Where: ″%1$s″ is the specifier in question.

Explanation: The ″%1$s″ specifier has been used on a declaration that is not in an appropriate scope.

User Response: Remove the specifier.

CCN5105 Functions declared within a block must not be ″%1$s″.

Where: ″%1$s″ is the specifier in question.

Explanation: A function declared in a lexical block scope cannot have the ″%1$s″ specifier.

User Response: Remove the specifier.

CCN5106 The ″static″ specifier must be applied only to objects, functions, and anonymous unions, structs and classes.

Explanation: The ″static″ specifier has been applied to an inappropriate object.

User Response: Remove the specifier.

CCN5107 The ″extern″ specifier must be applied only to objects and functions.

Explanation: The ″extern″ specifier cannot be applied to an out-of-line member variable or a type.

User Response: Remove the ″extern″ specifier.

CCN5108 Class members must not be declared ″extern″.

Explanation: The ″extern″ specifier cannot be applied to an out-of-line member variable.

User Response: Remove the ″extern″ specifier.

CCN5109 The ″mutable″ specifier must be applied only to class data members.

Explanation: The ″mutable″ specifier is being applied to a declaration that is not a member of a class.

User Response: Remove the ″mutable″ specifier.

CCN5110 The ″inline″ specifier must be applied only to function declarations.

Explanation: The ″inline″ specifier is being applied to something other than a function.

User Response: Remove the ″inline″ specifier.

CCN5111 The ″explicit″ specifier must be applied only to declarations of constructors within a class declaration.

Explanation: The ″explicit″ specifier is being applied to something other than a constructor that is being declared in-line in the class.

User Response: Remove the ″explicit″ specifier.

CCN5112 The ″virtual″ specifier must be applied only to declarations of non-static class member functions within a class declaration.

Explanation: An attempt is being made to apply the ″virtual″ specifier inappropriately.

User Response: Remove the ″virtual″ specifier from member functions using classes that are not static, or do not use it outside of a class.

CCN5113 The ″static″ specifier must be applied only to class member declarations within a class declaration.

Explanation: An attempt is being made to apply the ″static″ specifier inappropriately.

User Response: Remove the ″static″ specifier.

CCN5114 A parameter name must not be the same as another parameter of this function.

Explanation: All parameter names for a given function must be unique.

User Response: Give the parameter a unique name.

CCN5115 A member variable must have the ″%1$s″ attribute to be initialized in the definition of a class.

Where: ″%1$s″ is the missing specifier.

Explanation: Only constants that are also static may be initialized in the definition of a class.

User Response: Remove the initializer or ensure that the member is specified as both static and const.

CCN5116 A template declaration must declare a function, a class, a static member of a template class, or a template member of a class.

Explanation: An attempt is being made to create an invalid template.

User Response: Change the declaration so it is not a template, or correct the template declaration.

CCN5117 Linkage specification must be at namespace scope.

Explanation: Linkage specifications are only valid for declarations at namespace scope.

User Response: Remove the linkage specification.

CCN5118 A class name is expected in the base specifier.

Explanation: The name given in the base specifier is not a class.

User Response: Remove the base specifier or change it to refer to a class.

CCN5119 A friend template must not be declared in a local class.

Explanation: A friend of a class defined in a lexical block must not be a template.

User Response: Move the class to namespace scope or remove the friend declaration.

CCN5120 The out-of-line member definition ″%1$s″ of an explicit specialization should not use a template prefix.

Where: ″%1$s″ is the identifier of the out-of-line member.

Explanation: Out-of-line members of explicit specializations are defined in the same manner as members of non-template classes.

User Response: Remove the template prefix.

CCN5121 A template cannot have ″C″ linkage.

Explanation: Any linkage other than C++ is defined by implementation. The behavior with any linkage other than C++ is implementation-defined.

User Response: Remove the ″C″ linkage.

CCN5122 The duplicate attribute ″%1$s″ is ignored.

Where: ″%1$s″ is the duplicate attribute.

Explanation: The attribute ″%1$s″ has been specified more than once.

User Response: Remove the extra attributes.

CCN5123 The operator symbol is not recognized.

Explanation: The operator symbol specified is not valid.

User Response: Change the operator symbol to a valid symbol.

CCN5124 The text ″typename″ is unexpected because it cannot be used to modify a base specifier.

Explanation: A name specified in a base specifier list must be a type so typename is not required for template dependent names in a base specifier list.

User Response: Remove the ″typename″ elaboration from the name.

CCN5125 The duplicate specifier ″%1$s″ is ignored.

Where: ″%1$s″ is the duplicate specifier.

Explanation: The specifier ″%1$s″ has been specified more than once.

User Response: Remove the extra specifiers.

CCN5126 A template defined in an unnamed namespace must not be exported.

Explanation: Exported namespace scope template definitions must be in a named namespace.

User Response: Don't export the template, give the namespace a name, or move the template to another namespace scope.

CCN5127 The text ″typename″ is unexpected because it cannot be used to modify a name in a constructor initializer list.

Explanation: A name specified in a constructor initializer list must be a member or a base class so typename is not required for template dependent names in a constructor initializer list.

User Response: Remove the ″typename″ elaboration from the name.

CCN5128 ″%1$s″ is an ambiguous qualifier.

Where: ″%1$s″ is the ambiguous qualifier.

Explanation: The qualifier ″%1$s″ is ambiguous since there is more than one name to which it resolves.

User Response: Add extra qualification to remove the ambiguity.

CCN5129 The qualifier ″%1$s″ is not defined in the current scope.

Where: ″%1$s″ is the unknown qualifier.

Explanation: The name being used as a qualifier has not been declared in a visible scope.

User Response: Change the qualifier to a name that has been declared.

CCN5130 ″%1$s″ is not declared.

Where: ″%1$s″ is the unknown name.

Explanation: The name ″%1$s″ is not declared in any visible scope.

User Response: Change the name to one that has been declared.

CCN5131 Only one calling convention can be specified here.

Explanation: More than one calling convention is being specified.

User Response: Remove the extra calling conventions.

CCN5132 The expected token ″%1$s″ was not found.

Where: ″%1$s″ is the expected token.

Explanation: The coompiler expected ″%1$s″, and it was not there.

User Response: Insert the expected token, or check tokens requiring matched pairs.

CCN5133 The attributes ″%1$s″ are not allowed.

Where: ″%1$s″ is the invalid attributes.

Explanation: The specifier or qualifier ″%1$s″ is incorrect on this type of declaration.

User Response: Remove the invalid attributes.

CCN5134 A function return type must not be a type definition. There may be a missing ″;″ after a ″}″.

Explanation: An attempt has been made to define a class in the return type of a function. This is usually caused by a missing ″;″ after the class definition.

User Response: Change the return type or ensure that a previous class definition has a ″;″ at the end of it.

CCN5135 The array bound cannot be zero.

Explanation: An array cannot be declared with zero elements.

User Response: Change the array bound.

CCN5136 A return type must not be specified for a constructor.

Explanation: Constructors cannot have return types. A member or member function that has the same name as the class is considered a constructor, even if it is ill-formed.

User Response: Remove the return type or rename the member.

CCN5137 The attribute ″%1$s″ is not allowed for a constructor.

Where: ″%1$s″ is the invalid attribute.

Explanation: A declaration of a constructor cannot have the ″%1$s″ attribute.

User Response: Remove the attribute.

CCN5138 The undefined template ″%1$s″ must not be explicitly instantiated.

Where: ″%1$s″ is the identifier of the undefined template.

Explanation: An explicit instantiation requires a definition.

User Response: Define the template or remove the explicit instantiation.

CCN5139 In the context of the forward declaration, the name ″%1$s″ must not be qualified.

Where: ″%1$s″ is the qualified name.

Explanation: A qualified name cannot be used in a forward declaration for a class.

User Response: Remove the qualifiers from the name.

CCN5140 The text ″%1$s″ is unexpected. ″%2$s″ may be undeclared, ambiguous, or may require ″typename″ qualification.

Where: ″%1$s″ is the symbol causing the syntax error. ″%2$s″ is the name that may be causing the error if it is expected to be a type.

Explanation: There is a syntax error in the declaration. A name may be expected to be a type that is unknown or ambiguous, or the type specified may be template-dependent and require typename qualification.

User Response: Remove the offending symbol, ensure that the name used as a type name is actually a type, or add typename qualification to the type.

CCN5141 The declaration ″%1$s″ must not become a function because of a template argument.

Where: ″%1$s″ is the declaration that is acquiring function type.

Explanation: Only a declaration that uses the syntactic form of a function can be a function.

User Response: Change the template argument, or change the declaration.

CCN5142 cv-qualifiers must not be added to a typedef of function type.

Explanation: The const and volatile qualifiers cannot be specified on a type where a typedef that refers to a function is used.

User Response: Remove the const or volatile specifiers.

CCN5143 The qualifier ″%1$s″ is not a class.

Where: ″%1$s″ is the invalid qualifier.

Explanation: A typedef that does not refer to a class is being used as a qualifier.

User Response: Change the qualifier to refer to a class.

CCN5144 A non-local declaration is not allowed in a function body.

Explanation: Only local declarations are allowed in a function body.

User Response: Change the declaration to be a local declaration, or move it to the correct scope.

CCN5145 The explicit instantiation ″%1$s″ of the class template does not match the primary template.

Where: ″%1$s″ is the explicit instantiation.

Explanation: If the primary template is a union, the explicit instantiation must be a union as well. If the primary template is a class, the explicit instantiation must be a class.

User Response: Make sure that the class keys match.

CCN5146 The keyword ″friend″ is not allowed for a non-function.

Explanation: The ″friend″ keyword can only be used to nominate classes or functions as friends.

User Response: Remove the ″friend″ keyword.

CCN5147 Friend declarations are allowed only in classes and structs.

Explanation: Friends allow access to protected and private members. Because only classes and structs have members, only classes and structs can have friend declarations.

User Response: Remove the friend declaration.

CCN5148 A friend declaration must not be an explicit specialization.

Explanation: An explicit specialization declaration must not be a friend declaration.

User Response: Remove the friend or change it so it is not an explicit specialization.

CCN5149 A template defined in an unnamed namespace must not be exported.

Explanation: Exported namespace scope template definitions must be in a named namespace.

User Response: Do not export the template, give the namespace a name, or move the template to another namespace scope.

CCN5150 A using declaration must not specify a template-id.

Explanation: You cannot specify a template ID in a using declaration.

User Response: Remove or change the using declaration.

CCN5151 A friend function that is qualified must not be defined.

Explanation: Only friend functions without qualification can be defined in the friend declaration.

User Response: Define the friend function in a different declaration.

CCN5152 A template dependent name that is a type must be qualified with ″typename″.

Explanation: The keyword ″typename″ is used to identify a name in a template as a type.

User Response: Add the keyword typename.

CCN5153 The friend declaration is not a class or function.

Explanation: The friend declaration must only nominate a function or class for friendship.

User Response: Remove the friend specifier.

CCN5154 A class, struct, or union must not be defined in a friend declaration.

Explanation: Only functions can be defined in friend declarations.

User Response: Define the friend in another declaration.

CCN5155 A template parameter must not be used in an elaborated type specifier.

Explanation: If the identifier in an elaborated type specifier resolves to a typedef or a template type parameter, it is ill-formed.

User Response: Remove the construct.

CCN5156 ″%1$s″ keyword is not supported on this platform. The keyword is ignored.

Where: ″%1$s″ is the ignored keyword.

Explanation: The keyword has no meaning for the current platform and is ignored.

User Response: Remove the keyword for this platform.

CCN5157 The text ″>″ is unexpected. It may be that this token was intended as a template argument list terminator but the name is not known to be a template.

Explanation: An unexpected ″>″ was seen. This situation can arise when a template name is misspelled and is thus interpreted as a variable name rather than a template.

User Response: Check that previous template names are correct.

CCN5158 The function has a syntax error.

Explanation: The compiler does not recognize the syntax you used. There may be a typing error in this function.

User Response: Fix the syntax error.

CCN5159 A storage class cannot be specified on a declaration directly contained in a linkage specification.

Explanation: This declaration is contained within a linkage specification and therefore cannot have a storage class.

User Response: Remove the storage class.

CCN5160 ″__thread″ is not allowed on a class.

Explanation: The ″__thread″ specifier cannot be used on a declaration for a class.

User Response: Remove the ″__thread″ specifier.

CCN5161 ″%1$s″ is already specified.

Where: ″%1$s″ is the name that has been already specified.

Explanation: The name has already been specified.

User Response: Remove the duplicate name.

CCN5162 ″__thread″ is not allowed on an enumeration.

Explanation: The ″__thread″ specifier cannot be used in a declaration for an enumeration.

User Response: Remove the ″__thread″ specifier.

CCN5163 The array bound must not be negative.

Explanation: An array cannot be declared with a negative number of elements.

User Response: Change the array bound.

CCN5164 The operator ″%1$s″ is ambiguous.

Where: ″%1$s″ is the ambiguous operator.

Explanation: The specified operator is ambiguous because it can resolve to more than one declaration.

User Response: Add more qualifiers to resolve the ambiguity.

CCN5165 Only a positive integral constant which is a power of 2 is allowed in the __align specifier.

Explanation: The __align specifier must have a power of two since these are the only boundaries that align with memory.

User Response: Change the integral constant to be a power of two.

CCN5166 The __align specifier can only be applied to the definition of an aggregate tag or the declaration of a global or static variable.

Explanation: The __align specifier has been applied to an inappropriate type of declaration.

User Response: Remove the __align specifier.

CCN5167 This message is no longer used.

CCN5170 The class name ″%1$s″ must already be declared.

Where: ″%1$s″ is the invalid name.

Explanation: A declaration must exist in order to use this class name.

User Response: Declare the class name.

CCN5173 ″{″ is expected.

Explanation: An opening brace is expected for the function or member list.

User Response: Add appropriate bracing.

CCN5178 An enumeration must not contain both a negative value and an unsigned value greater than LONG_MAX.

Explanation: An enumeration cannot contain both negative values and unsigned values greater than LONG_MAX because they cannot both be represented by the same type.

User Response: Remove the invalid enumerators.

CCN5179 The enumeration value is too large.

Explanation: The enumeration value cannot be represented because it is too large for the underlying type.

User Response: Remove the invalid enumeration value.

CCN5182 Enumerator expected.

Explanation: A valid enumerator for an enumerator is required in an enum declaration.

User Response: Fix the syntax error by adding an enumerator.

CCN5184 The ″{″ has no matching ″}″.

Explanation: There are not enough ″}″s in the source so some construct is not complete.

User Response: Add the appropriate number of ″}″s.

CCN5185 The ″%1$s″ linkage specifier must only be applied to a function or a pointer to a function.

Where: ″%1$s″ is the linkage specifier from the user's source code.

Explanation: The ″%1$s″ linkage specifier is being applied to something other than a function or pointer to function.

User Response: Remove the linkage specifier.

CCN5186 A ″;″ or ″,″ is expected following the initializer.

Explanation: An initialiser was incomplete.

User Response: Add ″;″ after the initialiser.

CCN5187 The ″(″ has no matching ″)″.

Explanation: There is an imbalance of left and right parentheses.

User Response: Ensure that each left parenthesis has a matching right parenthesis.

CCN5188 A ″)″ or ″,″ is expected following the initializer.

Explanation: The initializer is not properly formed.

User Response: Add the appropriate ending token to complete in the initializer.

CCN5189 Only static member variables of templates can be instantiated.

Explanation: A non-static data member of a template cannot be explicitly instantiated.

User Response: Remove the explicit instantiation, or explicitly instantiate the class.

CCN5190 A ″{″ must follow a constructor initializer.

Explanation: A body for the constructor must follow the constructor initializer list.

User Response: Add a body for the constructor.

CCN5191 A handler must be a compound statement.

Explanation: A catch handler must be a lexical block enclosed by ″{″ and ″}″.

User Response: Add a well-formed catch handler.

CCN5192 A ″{″ must follow a base specifier list.

Explanation: Only class definitions can have a base specifier list. All class definitions must include a member list.

User Response: Add a member list to the class definition.

CCN5193 A typedef name cannot be used in this context.

Explanation: Only actual class names, and not typedef names, can be used in elaborations.

User Response: Replace the typedef name with the class it represents.

CCN5194 The ″%1$s″ declaration must declare a function.

Where: ″%1$s″ is the declaration from the user's source code.

Explanation: An operator or conversion function name in a declaration can only be used in a function declaration.

User Response: Change the name in the declaration.

CCN5195 The initializer has a syntax error.

Explanation: The initializer is not well-formed.

User Response: Correct the syntax error in the initializer.

CCN5196 A friend declaration must not declare a partial specialization.

Explanation: The partial specialization of a template class cannot be declared in a friend declaration.

User Response: Remove the friend declaration or change it from a partial specialization.

CCN5197 The ″asm″ keyword declaration is not supported.

Explanation: Inserting inline assembler instructions using the ″asm″ declaration is not supported. It is ignored.

User Response: Remove the ″asm″ declaration.

CCN5198 The omitted keyword ″private″ is assumed for base class ″%1$s″.

Where: ″%1$s″ is the name of the base class which is assumed to be private.

Explanation: The access to the base class is not specified and is assume to be private.

User Response: Add either ″public,″ ″protected,″ or

″private″ to the base class specifier.

CCN5199 An explicit instantiation must specify only a template class instantiation name.

Explanation: An explicit instantiation cannot contain a class definition. It must have a template argument list.

User Response: Correct or remove the explicit instantiation.

CCN5200 The ″%1$s″ operator is not allowed between ″%2$s″ and ″%3$s″.

Where: ″%1$s″ is the operator. ″%2$s″ and ″%3$s″ are the operands.

Explanation: The ″%1$s″ operator cannot be used between the two specified expressions because the operator is not defined for the types of the expression.

User Response: Change the operator or one or both of the operands.

CCN5201 The ″%1$s″ operator is not allowed for type ″%2$s″.

Where: ″%1$s″ is the operator. ″%2$s″ is the operand.

Explanation: The ″%1$s″ operator cannot be used with the specified expression because the operator is not defined for the type of the expression.

User Response: Change the operator or the operand.

CCN5202 An expression of type ″%1$s″ is not allowed on the left side of ″%2$s%3$s″.

Where: ″%2$s%3$s″ are the operands. ″%1$s″ is the operator.

Explanation: The type of the expression on the left side of the operator is not correct.

User Response: Change the left operand.

CCN5203 The member expression ″.%1$s″ or ″->%1$s″ must be used with the function call operator ().

Where: where ″%1$s″ is the name of the member function.

Explanation: The member expression refers to a member function so it must be used with the function call operator.

User Response: Add the function call operator with the parameters required for the member function call.

CCN5204 An expression of type ″%1$s″ must not be followed by the function call operator ().

Where: where ″%1$s″ is the type of the name referenced with the function call operator ().

Explanation: Only functions can be followed by a function call operator ().

User Response: Remove the function call operator ().

CCN5205 An expression of type ″%1$s″ is not allowed where an rvalue is expected.

Where: ″%1$s″ is the type of the expression.

Explanation: The expression cannot be used in this situation since it has void type.

User Response: Change the expression.

CCN5206 An rvalue of type ″%1$s″ cannot be converted to an rvalue of type bool.

Where: ″%1$s″ is the type of expression.

Explanation: There is no valid conversion sequence for converting the expression to an expression of type bool.

User Response: Change the expression or provide a conversion sequence.

CCN5207 No common type found for operands with type ″%1$s″ and ″%2$s″.

Where: ″%1$s″ and ″%2$s″ are the types of the operands.

Explanation: There is no standard conversion sequence between the two types.

User Response: Define a conversion sequence between the two types.

CCN5208 The operand for ″%1$s″ is of type ″%2$s″ but a pointer-to-member type is required.

Where: ″%1$s″ is the operator. ″%2$s″ is the unexpected type.

Explanation: The operator is expecting a pointer-to-member as an operand but the operand is of type ″%2$s″.

User Response: Change the operand to be a pointer-to-member.

CCN5209 The result of this pointer-to-member operator must be the operand of the function call operator ().

Explanation: This expression is expected to be a function call.

User Response: Change the expression to be a function call.

CCN5210 ″%1$s″ is not a base class of ″%2$s″.

Where: ″%1$s″ is the problematic class. ″%2$s″ is the expected derived class.

Explanation: The class specified is not a base class, so the devirtualisation or destructor name is not valid.

User Response: Change the name to refer to a base class.

CCN5211 The array operator must have one operand that is a pointer to a complete type and an operand that is of integral type.

Explanation: Either the variable is not an array or pointer or the index is not an integral type.

User Response: Change the variable to be an array or pointer or the index to be an integer.

CCN5212 The operand of the ″%1$s″ operator must be an lvalue.

Where: ″%1$s″ is the operator.

Explanation: The operator expects an object as its operand.

User Response: Change the operand to be an object.

CCN5214 The conditional expression of a switch statement must be of integral or enumeration type.

Explanation: Integral types are all sizes of int and char as well as enumerations. A switch statement condition must have an integral type or something that can be converted to an integral type.

User Response: Modify the switch condition or use an if statement instead of a switch.

CCN5215 The wrong number of arguments have been specified for ″%1$s″.

Where: Where ″%1$s″ is the name of the function being called.

Explanation: When a function is called, the arguments are matched against the actual parameters in the function declaration. There must be the same number of arguments in the call as there are parameters in the declaration unless there are default arguments specified.

User Response: Verify the function declaration and provide the correct number of arguments in your call.

CCN5216 An expression of type ″%1$s″ cannot be converted to type ″%2$s″.

Where: ″%1$s″ is the type being converted from. ″%2$s″ is the type being converted to.

Explanation: To convert between types, the compiler uses a set of specific rules defined in the C++ language. In this case the compiler was unable to convert between the specified types.

User Response: Modify the expression so that the conversion can be made, or define a conversion function to do the conversion.

CCN5217 ″%1$s″ is not a member of ″%2$s″.

Where: ″%1$s″ is the name of the member you are attempting to access. ″%2$s″ is the name of the class.

Explanation: When using the . or -> operators to access a class member, the name after the operator must be a member of the class.

User Response: Verify with the class declaration to see that you are accessing a member.

CCN5218 The call does not match any parameter list for ″%1$s″.

Where: ″%1$s″ is the name of the function.

Explanation: The compiler will attempt to match the arguments in your function call against all functions defined with the name you are calling. It cannot match the number and types or arguments in your call with one of the declarations for the function.

User Response: Check the declaration of the function you want to call and modify your arguments so that they match.

CCN5219 The call to ″%1$s″ has no best match.

Where: ″%1$s″ is the name of the function being called.

Explanation: When a function is called, the compiler will check all the function declarations it has for the name you are calling. In this case, the compiler was unable to determine which one to call because there is not a single version that is a best match. The criteria for a best match is based on the types of the parameters and the conversions required to match them with the arguments in your call.

User Response: Check the declarations for functions with that name and modify your arguments so that the correct one can be matched.

CCN5220 The address of a bit-field cannot be taken.

Explanation: C++ language standards indicate that the & operator cannot be applied to bit-fields.

User Response: Change the bit-field to an array or remove the line which attempts to take the address of the bit-field.

CCN5221 The case expression must be an integral constant expression.

Explanation: Integral types are all sizes of int and char as well as enumerations. A case expression must be an integral constant expression which is an expression which results in an integral type.

User Response: Modify the expression so that it is an integral constant expression, or change the switch statement to an if statement.

CCN5222 The function must not have a return value.

Explanation: The function was declared with a return type of void, so it cannot have a return value specified.

User Response: Remove the return value, or modify the function declaration to return the required type.

CCN5223 A return value of type ″%1$s″ is expected.

Where: ″%1$s″ is the type of the expected return value.

Explanation: The function was declared with a specific return type, so it should return a value of that type.

User Response: Modify the return type to match the declaration, or modify the declaration.

CCN5224 The type name ″%1$s″ is used where a variable or function name is expected.

Where: ″%1$s″ is the type name.

Explanation: The expression was expected to be an object or function name but a type name was found.

User Response: Replace the type with an object or function name.

CCN5225 The initializer list has too many initializers.

Explanation: An initializer list should not have more initializers than the number of elements to initialize.

User Response: Remove some initializers or increase the number of elements to initialize.

CCN5226 The initializer must not be enclosed in braces.

Explanation: Only initializers for classes and arrays can have braces ″{″ and ″}″.

User Response: Remove the braces.

CCN5227 ″%1$s″ cannot be initialized with an initializer list.

Where: ″%1$s″ is the type that cannot be initialized with an initializer list.

Explanation: The specified type cannot be initialized with an initializer list in braces ″{″ and ″}″.

User Response: Verify that the type is one that may be used with an initializer list. References cannot be initialized with an initializer list.

CCN5228 A ″&″ must precede the qualified member ″%1$s″ to form an expression with type pointer-to-member.

Where: ″%1$s″ is the member.

Explanation: A non-static member of a class was referred to with a qualified name, but no object is specified.

User Response: Refer to an object.

CCN5229 The best viable function ″%1$s″ uses an ambiguous conversion sequence.

Where: ″%1$s″ is the overloaded function.

Explanation: The overloaded function that has the closest match requires a conversion where one of the steps has more than one valid choice.

User Response: Provide a closer matching overload for the function being called.

CCN5230 The overloaded function name is not used in a valid context.

Explanation: It is not valid to use an overloaded function here.

User Response: Use a non-overloaded function.

CCN5231 The array bound must be specified and must be a positive integral constant expression.

Explanation: Only the first array bound in a series of array bounds can be omitted when declaring a multi-dimensional array.

User Response: Add the missing array bounds.

CCN5232 The implicit constructor for ″%1$s″ initializes a const member.

Where: ″%1$s″ is the class.

Explanation: The class contains a const member which must be initialized so a constructor must be provided.

User Response: Provide a constructor.

CCN5233 The implicit constructor for ″%1$s″ initializes a reference member.

Where: ″%1$s″ is the class.

Explanation: The class contains a reference member which must be initialized so a constructor must be provided.

User Response: Provide a constructor.

CCN5234 The implicit constructor for ″%1$s″ initializes a member of class type with an ill-formed constructor.

Where: ″%1$s″ is the class.

Explanation: The class contains a member of class type which does not have a default constructor so a constructor must be provided.

User Response: Provide a constructor.

CCN5235 The implicit constructor for ″%1$s″ initializes a base class with an ill-formed constructor.

Where: ″%1$s″ is the class.

Explanation: The class has a base class which does not have a default constructor so a constructor must be provided.

User Response: Provide a constructor.

CCN5236 The constructor initializer is unexpected. All bases and members have been initialized.

Explanation: The constructor initialiser list has more elements being initialized than exist in the class. Either objects are initialized more than once or non-members are in the initializer list.

User Response: Remove the extra initializers from the constructor initializer list.

CCN5237 ″%1$s″ designates both a direct non-virtual base class and an inherited virtual base class.

Where: ″%1$s″ is the ambiguous base class name.

Explanation: The class ″%1$s″ is ambiguous because it refers to both a virtual base class and a non-virtual base class.

User Response: Add qualifiers to make the name unambiguous.

CCN5238 The data member ″%1$s″ cannot be initialized because there is no corresponding default constructor.

Where: ″%1$s″ is the class member.

Explanation: The data member was not in the constructor initializer list, but the type does not have a default constructor so the type cannot be constructed.

User Response: Add the member to the constructor initializer list.

CCN5239 The base class ″%1$s″ cannot be initialized because it does not have a default constructor.

Where: ″%1$s″ is the base class.

Explanation: The base class was not in the constructor initializer list. The type does not have a default constructor so the base class cannot be constructed.

User Response: Add the base class to the constructor initializer list.

CCN5240 A duplicate case value is not allowed.

Explanation: The switch statement cannot choose a single case if there are duplicate case values.

User Response: Remove or modify the duplicate case value.

CCN5241 A ″%1$s″ statement is not allowed in this scope.

Where: ″%1$s″ is the type of statement.

Explanation: It is not valid to have this type of statement in this scope.

User Response: Remove the statement.

CCN5242 ″goto %1$s″ bypasses the initialization of ″%2$s″.

Where: ″%1$s″ is the label. ″%2$s″ is the missed variable.

Explanation: The goto statement skips over the initialization of an automatic variable.

User Response: Move the label before the variable declaration.

CCN5243 Label ″%1$s″ is already defined.

Where: ″%1$s″ is the duplicate label.

Explanation: A label can only refer to one location in a function.

User Response: Rename the label.

CCN5244 Label ″%1$s″ is not declared in this function.

Where: ″%1$s″ is the missing label.

Explanation: Labels are only visible within the function in which they exist; either the label is not defined or it is in a different function than the goto.

User Response: Add the label to the function.

CCN5245 The switch statement already has a ″default″ statement.

Explanation: A switch statement may contain only one default statement.

User Response: Remove the extra default statement.

CCN5246 The ″%1$s″ statement bypasses the initialization of ″%2$s″.

Where: ″%1$s″ is the case or default statement. ″%2$s″ is the bypassed variable.

Explanation: A case for the switch contains automatic variables that are not contained within a compound statement.

User Response: Add a pair of braces {} to enclose the code containing the automatic variable.

CCN5247 This message is no longer used.

CCN5248 ″%1$s″ is not a class name.

Where: ″%1$s″ is the name.

Explanation: The name was expected to be a class name but it is not.

User Response: Change the name to be a class name.

CCN5249 Default arguments are not available due to other errors.

Explanation: This error is a cascade error. The default initialisers cannot be used because of other errors.

User Response: Fix the errors in the default initializers.

CCN5250 The keyword ″this″ is only allowed in a non-static class member function body or in a constructor member initializer.

Explanation: The ″this″ keyword has been used in the wrong context.

User Response: Remove the ″this″ keyword.

CCN5251 The ″%1$s″ operator cannot be applied to the undefined class ″%2$s″.

Where: ″%1$s″ is the operator. ″%2$s″ is the undefined class.

Explanation: The use of the ″%1$s″ operator requires that the class that is being used as the operand be defined and not just declared.

User Response: Define the class.

CCN5252 ″%1$s″ contains a circular reference back to ″%2$s″.

Where: ″%1$s″ and ″%2$s″ are the classes with circular references.

Explanation: The two classes contain references to each other that require each class to be defined before the other.

User Response: Change one of the classes so that it does not require the other class to be defined.

CCN5253 This use of undefined class ″%1$s″ is not valid.

Where: ″%1$s″ is the class.

Explanation: The usage requires that the class be defined and not just declared.

User Response: Define the class.

CCN5254 The non-static member ″%1$s″ must be associated with an object or a pointer to an object.

Where: ″%1$s″ is the member.

Explanation: A member of a class has been referred to without an object but it is not a static member.

User Response: Specify an object.

CCN5255 The implicit member function ″%1$s″ cannot be defined.

Where: ″%1$s″ is the member function that cannot be defined.

Explanation: This is a cascading error. The implicit member function cannot be defined due to other errors in the class.

User Response: Fix the errors in the class.

CCN5256 A parameter of type ″%2$s″ cannot be initialized with an expression of type ″%1$s″.

Where: ″%2$s″ is the parameter type. ″%1$s″ is the initialization expression type.

Explanation: The type of the argument for the function does not match the type of the parameter.

User Response: Change the type of the parameter to match the expected type.

CCN5257 An object or reference of type ″%2$s″ cannot be initialized with an expression of type ″%1$s″.

Where: ″%2$s″ is the object or reference type. ″%1$s″ is the initialization expression type.

Explanation: The type of the expression is not correct for initializing the object or reference.

User Response: Change the type of the initializer.

CCN5258 A return value of type ″%2$s″ cannot be initialized with an expression of type ″%1$s″.

Where: ″%2$s″ is the return value type. ″%1$s″ is the initialization expression type.

Explanation: The type of the expression in the return statement does not match the return type of the function.

User Response: Change the type of the expression to the return type of the function.

CCN5259 The name lookups of ″%1$s″ do not yield the same type in the context of the expression and in the context of the class of the object expression.

Where: ″%1$s″ is the name being looked up.

Explanation: When a qualified name is specified in a member access, it is looked up in the context specified on the left side of the ″.″ or ″->″ and in the context of the entire expression. It must resolve in only one of these lookups or it must resolve to the same declaration in both lookups.

User Response: Change the name.

CCN5260 A goto must not enter a try block or handler.

Explanation: A goto has been specified to a label that is in a try block or catch handler that does not also contain the goto statement.

User Response: Change the label.

CCN5261 The header <typeinfo> must be included before using the typeid operator.

Explanation: The use of the typeid operator requires that the standard header <typeinfo> be included using a #include directive before it is used.

User Response: Include the <typeinfo> header.

CCN5262 The first argument to the ″offsetof″ macro must be a class type.

Explanation: The ″offsetof″ macro can only be used with class types.

User Response: Change the first argument to be a class type.

CCN5263 The non-const member function ″%1$s″ is called for ″%2$s″.

Where: ″%1$s″ is the function. ″%2$s″ is the object.

Explanation: Only const member functions can be called with a const object.

User Response: Change the member function to be const or change the object to be non-const.

CCN5264 The non-volatile member function ″%1$s″ is called for ″%2$s″.

Where: ″%1$s″ is the function. ″%2$s″ is the object.

Explanation: Only volatile member functions can be called with a volatile object.

User Response: Change the member function to be volatile or change the object to be non-volatile.

CCN5265 A pointer to non-const member function type ″%1$s″ is called for ″%2$s″.

Where: ″%1$s″ is the function. ″%2$s″ is the type.

Explanation: Only const member functions can be called with a const pointer-to-member.

User Response: Change the member function to be const or change the pointer-to-member to be const.

CCN5266 A pointer to non-volatile member function type ″%1$s″ is called for ″%2$s″.

Where: ″%1$s″ is the function. ″%2$s″ is the type.

Explanation: Only volatile member functions can be called with a volatile pointer-to-member.

User Response: Change the member function to be volatile or change the pointer-to-member to be volatile.

CCN5267 The second operand to the ″offsetof″ macro is not valid.

Explanation: The second operand of the ″offsetof″ macro is expected to be a member.

User Response: Change the second operand to be a member.

CCN5268 ″%1$s″ has more than one default constructor.

Where: ″%1$s″ is the class.

Explanation: A class can only have one default constructor. A constructor with default initializers for all but the first parameter is considered a default constructor if all of the defaults are used.

User Response: Remove one of the default initializers or specify more arguments when calling the constructor.

CCN5269 ″%1$s″ has no default constructor.

Where: ″%1$s″ is the class.

Explanation: The class has no default constructor and one cannot be generated since the class contains objects that do not have default constructors.

User Response: Specify a default constructor.

CCN5270 An object of type ″%2$s″ cannot be constructed from an lvalue of type ″%1$s″.

Where: ″%2$s″ and ″%1$s″ are the types of the target and the expression.

Explanation: There is no constructor for the object that can be used for constructing the object.

User Response: Add an appropriate constructor or change the type.

CCN5271 ″%1$s″ is an ambiguous base class of ″%2$s″.

Where: ″%1$s″ is the base. ″%2$s″ is the class.

Explanation: The base class is ambiguous because the class has more than one base class with the same name.

User Response: Add qualifiers to uniquely specify the base class.

CCN5272 An array allocated by ″new″ cannot have an initializer.

Explanation: An initializer cannot be specified for an array that is allocated using new.

User Response: Remove the initializer.

CCN5273 The array bound must have a positive value.

Explanation: An array cannot be declared with a negative number of elements.

User Response: Change the array bound.

CCN5274 The name lookup for ″%1$s″ did not find a declaration.

Where: ″%1$s is the unresolved name.

Explanation: The name is not declared within this or an enclosing scope.

User Response: Declare the variable or change the name.

CCN5275 The array boundary must have integral type or enumeration type.

Explanation: Only integral types can be used to specify an array bound.

User Response: Change the array bound to be an integral type.

CCN5276 The local variable ″%1$s″ cannot be used in this context.

Where: ″%1$s″ is the local variable.

Explanation: A local variable cannot be used to specify default initializers for a function.

User Response: Remove the default initializers.

CCN5277 The local variable ″%1$s″ from function ″%2$s″ cannot be used in function ″%3$s″.

Where: ″%1$s″ is the variable. ″%2$s″ is the enclosing function. ″%3$s″ is the current function.

Explanation: A local variable from an enclosing function cannot be used in this context.

User Response: Remove the variable usage.

CCN5278 The reference variable ″%1$s″ must be initialized.

Where: ″%1$s″ is the reference variable.

Explanation: All reference variables must be initialized but no initializer is specified.

User Response: Specify an initializer.

CCN5279 The class member ″%1$s″ of type ″%2$s″ must be initialized in the initializer list of the constructor.

Where: ″%1$s″ is the member. ″%2$s″ is the class type.

Explanation: The member must be initialized in the constructor initializer list.

User Response: Add an initializer to the constructor initializer list.

CCN5280 The initializer is too long.

Explanation: The initializer for the array has too many initializers.

User Response: Remove the extra initializers.

CCN5281 An expression of type ″%1$s″ cannot be modified.

Where: ″%1$s″ is the type that cannot be modified.

Explanation: The expression on the left side of the assignment or reference parameter cannot be modified.

User Response: Substitute an object that can be modified.

CCN5282 The const variable ″%1$s″ is uninitialized.

Where: ″%1$s″ is the const variable.

Explanation: All const variables must be initialized.

User Response: Initialize the variable.

CCN5283 ″%1$s″ is not a valid type for a function-style cast.

Where: ″%1$s″ is the type that is attempting to be cast to.

Explanation: Only simple type specifiers (built-in types and named types) can be used in a function-style cast.

User Response: Change the type of the cast.

CCN5284 The bit-field ″%1$s″ cannot be bound to a non-const reference.

Where: ″%1$s″ is the bit-field.

Explanation: A bit-field can only be bound to a non-volatile const reference.

User Response: Change the reference type.

CCN5285 The expression calls the undefined pure virtual function ″%1$s″.

Where: ″%1$s″ is the function.

Explanation: Undefined pure virtual functions cannot be directly called.

User Response: Change the function being called.

CCN5286 The unqualified member ″%1$s″ should be qualified with ″%2$s::″ and preceded by an ″&″ when forming an expression with type pointer-to-member.

Where: ″%1$s″ is the member. ″%2$s″ are the qualifiers.

Explanation: A non-static member must be associated with an object.

User Response: Add the qualifiers and address operator.

CCN5287 ″offsetof″ must not be applied to ″%1$s″. It is not a POD (plain old data) type.

Where: ″%1$s″ is the type.

Explanation: ″offsetof″ cannot be applied to a class that is not a POD. POD types do not have non-static pointers-to-member, non-POD members, destructors, or copy assignment operators (that is, they are similar to C-style structs).

User Response: Change the type to be a POD type.

CCN5288 The function template parameter of type ″%2$s″ cannot be initialized with an argument of type ″%1$s″.

Where: ″%2$s″ is the function template parameter type. ″%1$s″ erroneous argument specified.

Explanation: The type of the argument is not appropriate for the type expected.

User Response: Change the type of the argument.

CCN5289 The function template parameter ″%1$s″ has been found to have two types: type ″%2$s″ and type ″%3$s″.

Where: ″%1$s″ is the template parameter. ″%2$s″ and ″%3$s″ are the two conflicting deduced types.

Explanation: Template argument deduction has arrived at two equally likely types for the same template type parameter.

User Response: Explicitly specify the template arguments.

CCN5290 The function template parameter ″%1$s″ has been found to have two values: ″%2$s″ and ″%3$s″.

Where: ″%1$s″ is the template parameter. ″%2$s″ and ″%3$s″ are the two conflicting deduced values.

Explanation: Template argument deduction has arrived at two equally likely values for the same non-type template parameter.

User Response: Explicitly specify the template arguments.

CCN5291 The template argument for ″%1$s″ cannot be found.

Where: ″%1$s″ is the template parameter.

Explanation: Template argument deduction has failed. Either nothing matched or there was an ambiguity.

User Response: Explicitly specify the template argument, or change the template.

CCN5292 This message is no longer used.

CCN5293 The argument to va_start must be a parameter name.

Explanation: A non-parameter has been specified to va_start.

User Response: Change the argument to a parameter name.

CCN5294 An object or reference of type ″%2$s″ cannot be initialized with an rvalue of type ″%1$s″.

Where: ″%2$s″ is the type of the object. ″%1$s″ is the type of the rvalue.

Explanation: This object or reference must be initialized with an object.

User Response: Change the type of the object or reference.

CCN5295 A parameter of type ″%2$s″ cannot be initialized with an rvalue of type ″%1$s″.

Where: ″%2$s″ is the type of the parameter. ″%1$s″ is the type of the rvalue.

Explanation: This parameter must be initialized with an object.

User Response: Change the type of the parameter.

CCN5296 A return value of type ″%2$s″ cannot be initialized with an rvalue of type ″%1$s″.

Where: ″%2$s″ is the return type. ″%1$s″ is the type of the rvalue.

Explanation: The return value must be initialized with an object.

User Response: Change the return type.

CCN5297 This message is no longer used.

CCN5298 Template argument deduction cannot be performed using the function ″%1$s″.

Where: ″%1$s″ is the name of the function.

Explanation: Argument deduction can only be performed with a function if the set of overloaded functions does not contain a template function.

User Response: Explicitly specify the template argument or change the template.

CCN5299 The ″%1$s″ operator cannot be applied to a pointer to incomplete type: ″%2$s″.

Where: ″%1$s″ is the operator. ″%2$s″ is the incomplete type.

Explanation: The ″%1$s″ operator requires that the type of its operand be defined and not just declared.

User Response: Define the type of the operand.

CCN5300 The ″private″ member ″%1$s″ cannot be accessed.

Where: ″%1$s″ is the member.

Explanation: The member is declared in a private section of the class and cannot be accessed.

User Response: Change the access of the member.

CCN5301 The ″protected″ member ″%1$s″ cannot be accessed.

Where: ″%1$s″ is the member.

Explanation: The member is declared in a protected section of the class and cannot be accessed.

User Response: Change the access of the member or remove the reference.

CCN5302 ″%1$s″ is a ″private″ base class of ″%2$s″.

Where: ″%1$s″ is the base class. ″%2$s″ is the derived class.

Explanation: The base class is private and cannot be accessed.

User Response: Change the access of the base class.

CCN5303 ″%1$s″ is a ″protected″ base class of ″%2$s″.

Where: ″%1$s″ is the base class. ″%2$s″ is the derived class.

Explanation: The base class is protected and cannot be accessed.

User Response: Change the access of the base class.

CCN5304 The ″private″ copy constructor ″%1$s″ cannot be accessed to create a temporary object.

Where: ″%1$s″ is the copy constructor.

Explanation: The creation of a temporary object requires access to the copy constructor, but the copy constructor is private.

User Response: Change the access of the copy constructor.

CCN5305 The ″protected″ copy constructor ″%1$s″ cannot be accessed to create a temporary object.

Where: ″%1$s″ is the copy constructor.

Explanation: The creation of a temporary object requires access to the copy constructor, but the copy constructor is protected.

User Response: Change the access of the copy constructor.

CCN5306 The ″private″ copy constructor ″%1$s″ cannot be accessed.

Where: ″%1$s″ is the copy constructor.

Explanation: Access to the copy constructor is required but the copy constructor is private.

User Response: Change the access of the copy constructor.

CCN5307 The ″protected″ copy constructor ″%1$s″ cannot be accessed.

Where: ″%1$s″ is the copy constructor.

Explanation: Access to the copy constructor is required but the copy constructor is protected.

User Response: Change the access of the copy constructor.

CCN5308 The semantics specify that a temporary object must be constructed.

Explanation: Informational message indicating that the semantics of the language require a temporary object to be constructed.

User Response: See the primary message.

CCN5309 The temporary is not constructed, but the copy constructor must be accessible.

Explanation: Informational message that the temporary is not constructed as an optimization but the language semantics require that the copy constructor be accessible.

User Response: See the primary message.

CCN5310 The assignment-style initialization of an object of type ″%1$s″ with an expression of type ″%2$s″ requires access to the copy constructor.

Where: ″%1$s″ is the type of the object. ″%2$s″ is the type of the expression.

Explanation: An assignment-style initialization requires access to the copy constructor, but the parentheses-style initialization does not.

User Response: Make the assignment operator a friend of the class or use parenthesis-style initialization.

CCN5311 Access to the copy constructor is not required if parentheses-style initialization is used.

Explanation: An assignment-style initialization requires access to the copy constructor, but the parentheses-style initialization does not.

User Response: Make the assignment operator a friend of the class or use parenthesis-style initialization.

CCN5400 ″%1$s″ has a conflicting declaration.

Where: ″%1$s″ is the name which has a conflicting declaration

Explanation: The specified name has already been given a different declaration.

User Response: Change the name for this declaration, or use the existing declaration.

CCN5401 The member ″%1$s″ is already declared.

Where: ″%1$s″ is the name of the member.

Explanation: The member name has already been used in this class. The compiler cannot tell the difference between two members with the same name unless they are both member functions with different parameters.

User Response: Change the name of the member, or use the existing declaration. If the member name is a member function, modify the parameters to overload the function.

CCN5402 The non-static member ″%1$s″ must not be defined outside of the class definition.

Where: ″%1$s″ is the member.

Explanation: Only static members can have a definition outside of the class definition. Non-static members only exist when a object is created from the class.

User Response: Move the definition of the member inside the class constructor or make the member static.

CCN5403 ″%1$s″ is already defined.

Where: ″%1$s″ is the name which has already been defined.

Explanation: The specified name has already been defined in another location.

User Response: Remove one of the definitions for this name, or use another name.

CCN5404 The out-of-line member function declaration for ″%1$s″ must have a body.

Where: ″%1$s″ is the name of the member function.

Explanation: A member function must be declared inside its class and may be defined either inside its class or outside its class. It may not be redeclared outside its class.

User Response: Add the definition for the body of this function.

CCN5405 The default arguments for ″%1$s″ must not be redefined.

Where: ″%1$s″ is the name of the function.

Explanation: If there is more than one declaration for the specified function, the default arguments should be given the same values in both.

User Response: Remove the duplicate declaration, or change the default arguments so that they match.

CCN5406 The namespace alias ″%1$s″ is already defined.

Where: ″%1$s″ is the namespace alias.

Explanation: A namespace alias in a declarative region can only be redefined to denote the same namespace.

User Response: Remove or change the namespace alias.

CCN5407 The base class ″%1$s″ contains a circular reference back to ″%2$s″.

Where: ″%1$s″ and ″%2$s″ are the names of the conflicting classes.

Explanation: A reference in the base class requires that the derived class be complete. There is no way to complete both classes.

User Response: Change one of the classes to remove the circularity.

CCN5408 The base class ″%1$s″ is declared but not defined.

Where: ″%1$s″ is the name of the base class.

Explanation: A base class must be a complete class.

User Response: Define the base class before it is used in a base specifier list.

CCN5409 ″%1$s″ must not be used more than once in the list of base classes.

Where: ″%1$s″ is the name of the duplicate base class.

Explanation: Listing the same class twice or more in a base specifier list is not allowed.

User Response: Remove the duplicate base class.

CCN5410 The direct base ″%1$s″ of class ″%2$s″ is ignored because ″%1$s″ is also an indirect base of ″%2$s″.

Where: ″%1$s″ is the name of the base class. ″%2$s″ is the name of the derived class.

Explanation: The base class has been specified directly as well as indirectly.

User Response: None needed, but the redundant base class can be removed.

CCN5411 The default arguments for ″%1$s″ are in error.

Where: ″%1$s″ is the template parameter declaration.

Explanation: A default template argument cannot refer to the template parameter.

User Response: Correct the default arguments.

CCN5412 The union ″%1$s″ cannot be used as a base class.

Where: ″%1$s″ is the name of the union.

Explanation: A union must not have, or be used as a base class.

User Response: Remove the union base specifier or change it to a class.

CCN5413 ″%1$s″ is already declared with a different access.

Where: ″%1$s″ is the name of the member.

Explanation: A member declaration must have only one access.

User Response: Remove the offending declaration or declare it with the same access.

CCN5414 ″%1$s″ is declared differently in the body of function ″%2$s″.

Where: ″%1$s″ is the duplicate local declaration. ″%2$s″ is the function containing it.

Explanation: The specified local name has already been given a different declaration.

User Response: Change the name for this declaration, or remove the conflicting duplicate declaration.

CCN5415 ″%1$s″ is already declared with default template arguments.

Where: ″%1$s″ is the name of the template parameter.

Explanation: A template parameter may not be given default arguments in two different declarations.

User Response: Remove the default argument on one of the declarations.

CCN5416 ″%1$s″ cannot be declared because its name has already been used.

Where: ″%1$s″ is the member name.

Explanation: A member can only be declared once in a class.

User Response: Change or remove one of the uses.

CCN5417 The qualified id-declarator ″%1$s″ cannot refer to a name introduced by a using declaration.

Where: ″%1$s″ is the qualified ID.

Explanation: The qualified ID collides with a name in a using declaration.

User Response: Change the declaration or remove the using declaration.

CCN5418 The definition of ″%1$s″ cannot contain an initializer because the initializer was specified in the class definition.

Where: ″%1$s″ is the data member.

Explanation: The out-of-line definition of a static data member can only have an initializer when there is no initializer on the declaration in the class.

User Response: Remove one of the initializers.

CCN5419 An exception-specification must be specified as ″%1$s″ to match the implicit declaration.

Where: ″%1$s″ is the exception specification.

Explanation: All declarations of a function including definitions and explicit specializations must have either no exception specification or the same set of types listed in their exception specifications.

User Response: Correct the exception specification.

CCN5420 ″%1$s″ is declared differently than the implicit declaration ″%2$s″.

Where: ″%1$s is the declaration. ″%2$s″ is the implicit declaration.

Explanation: A duplicate declaration of an implicit declaration is in error.

User Response: Correct or remove the declaration.

CCN5421 ″%1$s″ is declared differently than the internally generated declaration ″%2$s″.

Where: ″%1$s is the declaration. ″%2$s″ is the internally generated declaration.

Explanation: A duplicate declaration of an internal declaration is in error.

User Response: Correct or remove the declaration.

CCN5422 ″%1$s″ cannot be declared before ″%2$s″, and ″%2$s″ cannot be declared before ″%1$s″.

Where: ″%1$s″ and″%2$s″ are the two declarations.

Explanation: Each of the two declarations is coded so that it requires the other declaration first.

User Response: Change the dependence between the two declarations.

CCN5423 The new declaration ″%1$s″ cannot be added.

Where: ″%1$s″ is the declaration.

Explanation: The IDE is browsing and can't add a new declaration to the code store.

User Response: Reincorporate with the changed source.

CCN5424 ″%1$s″ is declared on line %3$s of ″%2$s″.

Where: ″%1$s″ is the declaration. %3$s is the line number. ″%2$s″ is the source.

Explanation: An informational message message giving the location of a declaration.

User Response: See the primary message.

CCN5425 ″%1$s″ is defined on line %3$s of ″%2$s″.

Where: ″%1$s″ is the declaration. %3$s is the line number. ″%2$s″ is the source.

Explanation: An informational message message giving the location of a definition.

User Response: See the primary message.

CCN5426 The name ″%1$s″ is used on line %3$s of ″%2$s″.

Where: ″%1$s″ is the name. %3$s is the line number. ″%2$s″ is the source.

Explanation: An informational message message giving the location of the use of a name.

User Response: See the primary message.

CCN5427 The using declaration introduces ″%1$s″ in conflict with a declaration in this scope.

Where: ″%1$s″ is the declaration in conflict.

Explanation: A using declaration is a declaration, so the restrictions on declaring the same name twice in the same region apply.

User Response: Remove the using declaration or remove the conflicting declaration.

CCN5428 The using declaration ″%1$s″ must not introduce a name into its own scope.

Where: ″%1$s″ is the using declaration.

Explanation: A using declaration is a declaraton, so the restrictions on declaring the same name twice in the same region apply.

User Response: Remove or change the using declaration.

CCN5429 ″%1$s″ must not be repeated at block scope.

Where: ″%1$s″ is the using declaration.

Explanation: A using declaration is a declaration, so the restrictions on declaring the same name twice in the same region apply (a variable at lexical block scope in this case).

User Response: Remove the repeated using declaration.

CCN5430 The out-of-line member declaration for ″%1$s″ must be in a namespace scope that encloses the class definition.

Where: ″%1$s″ is the out-of-line member declaration.

Explanation: The class definition cannot be seen in the scope that the out-of-line member declaration exists.

User Response: Move the out-of-line member declaration into the same scope as its class definition or a scope that encloses its class definition.

CCN5431 The declarator cannot be qualified with the enclosing namespace ″%1$s″.

Where: ″%1$s″ is the namespace declaration.

Explanation: A nested-name-specifier cannot name any of the namespaces that enclose the member's definition.

User Response: Remove the qualifiers.

CCN5432 The qualified declarator ″%1$s″ must refer to an existing declaration.

Where: ″%1$s″ is the qualified declarator.

Explanation: When the declarator-id is qualified, the declaration has to refer to a previously declared member of a class or namespace and the member cannot have been introduced by a using declaration already.

User Response: Remove the qualified ID, or add it to the class or namespace.

CCN5433 The explicitly specialized template class member ″%1$s″ cannot be defined unless the template class is specialized.

Where: ″%1$s″ is the explicitly specialized template class member.

Explanation: An out-of-line class member definition can only be made for an existing class. A class template explicit specialization is a separate class with different members from the primary template.

User Response: Write the class template explicit specialization or remove this declaration.

CCN5434 The friend function must also be declared in the enclosing block scope.

Explanation: If a friend declaration appears in a local class and the name specified is an unqualified name, a prior declaration is looked up without considering scopes that are outside the innermost enclosing non-class scope. For a friend function declaration, if there is no prior declaration, the program is ill-formed.

User Response: Remove the local friend function or add the declaration to the enclosing block scope.

CCN5435 The template ″%1$s″ must not be explicitly specialized more than once with the same set of template arguments.

Where: ″%1$s″ is the template.

Explanation: This is a violation of the one definition rule.

User Response: Remove the duplicate explicit specialization.

CCN5436 The template ″%1$s″ must not be explicitly instantiated more than once with the same set of template arguments.

Where: ″%1$s″ is the template.

Explanation: Only one explicit instantiation of a template with the same set of arguments is allowed in a program.

User Response: Remove the duplicate explicit instantiation.

CCN5437 The template ″%1$s″ must not be explicitly specialized and explicitly instantiated with the same set of template arguments.

Where: ″%1$s″ is the template.

Explanation: A program can have either explicit instantiation or explicit specialization of a template with the same set of arguments, but not both.

User Response: Remove either the explicit specialization or the explicit instantiation.

CCN5438 The template parameter ″%1$s″ must not be redeclared.

Where: ″%1$s″ is the template parameter.

Explanation: A template parameter can be declared at most once in a template parameter list.

User Response: Remove or change the template parameter.

CCN5439 The template parameters ″%1$s″ do not match the parameters for the previous declaration for ″%2$s″.

Where: ″%1$s″ and ″%2$s″ are the template parameters.

Explanation: A redeclaration of a template must agree in the number and type of the template parameters.

User Response: Correct the template parameters.

CCN5500 The configuration file ″%1$s″ cannot be opened: %2$s.

Where: ″%1$s″ is the name of the configuration file that could not be opened. ″%2$s″ is the string returned by the operating system when the file open failed.

Explanation: The configuration file could not be opened.

User Response: Check the permissions on the configuration file and that it exists.

CCN5501 The directive in the configuration file is not recognized.

Explanation: The directive in the configuration file is not recognized.

User Response: Change the directive.

CCN5502 The build was interrupted.

Explanation: The compilation was interupted and stopped.

User Response: Start the compile again.

CCN5503 The name is already used in the configuration file.

Explanation: The identifier has already been used in the configuration file.

User Response: Change the name to be another name that is not already used.

CCN5504 The template argument must be a constant integral expression.

Explanation: The argument for the template was not an integral constant expression.

User Response: Change the expression to be an integral constant expression.

CCN5505 The build failed and there are no messages.

Explanation: The compiler has experienced an internal failure.

User Response: Report the problem to your IBM C++ service representative.

CCN5506 The configuration file ″%1$s″ is empty.

Where: ″%1$s″ is the name of the configuration file.

Explanation: The configuration file is empty.

User Response: Check that the right configuration file has been specified.

CCN5507 The attempt to load %1$s from the default library path failed.

Where: ″%1$s″ is the name of the extension that failed to load.

Explanation: The dynamic load of the compiler extension failed.

User Response: Check the tool option on the command line or in the configuration file.

CCN5508 The file ″%1$s″ cannot be loaded: the program file is not an ordinary file, or its mode does not allow execution, or search permission is denied on a component of the path prefix.

Where: ″%1$s″ is the name of the file.

Explanation: The loading of the file failed because of access permissions or it was incorrectly specified.

User Response: Check the tool option on the command line or in the configuration file.

CCN5509 The file ″%1$s″ cannot be loaded: the program file has a valid magic number in its header, but the header is damaged or is incorrect for the machine on which the file is to be run.

Where: ″%1$s″ is the name of the file.

Explanation: The program could not be loaded because the header for the file is corrupt.

User Response: Ensure that the file has not been corrupted.

CCN5510 The file ″%1$s″ cannot be loaded: too many symbolic links were encountered in translating the path name.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because there were too many symbolic links in the path name.

User Response: Remove some of the symbolic links in the path name.

CCN5511 The file ″%1$s″ cannot be loaded: incorrect XCOFF header or some problems in linking.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because the header is corrupt or improperly linked.

User Response: Ensure that the file has not been corrupted.

CCN5512 The file ″%1$s″ cannot be loaded: the program requires more memory than is allowed by the system.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because it requires too much memory.

User Response: Increase the allocated memory to the program.

CCN5513 The file ″%1$s″ cannot be loaded: the file is currently open for writing by a process.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because it is currently open for writing.

User Response: Ensure that the file is not being used by another process and recompile.

CCN5514 The file ″%1$s″ cannot be loaded: a component of a path name exceeded 255 characters, or an entire path name exceeded 1023 characters.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because the path or some component of the path is too long.

User Response: Shorten the length of the path or of the component of the path that is too long.

CCN5515 The file ″%1$s″ cannot be loaded: a component of the file name does not exist.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because some component of the name does not exist.

User Response: Ensure that all directories in the path name exist or change the path for the file.

CCN5516 The file ″%1$s″ cannot be loaded: a component of the path prefix is not a directory.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because one of the components of the name is not a directory.

User Response: Change the path so that all components in the path prefix are directories.

CCN5517 The file ″%1$s″ cannot be loaded: the process root or current directory is located in a virtual file system that has been unmounted.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because the file system is not mounted.

User Response: Mount the required file system.

CCN5518 The file ″%1$s″ cannot be loaded: the file name is null.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because the file name is null.

User Response: Ensure that the file name is not null.

CCN5519 The file ″%1$s″ cannot be loaded: the file cannot be found.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because the could not be found.

User Response: Ensure that the file exists.

CCN5522 The file ″%1$s″ cannot be loaded: DosLoadModule return code is %2$s.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because of operating system errors.

User Response: Ensure that the file is correctly specified for the operating system.

CCN5523 Linkage %1$s is not known. extern ″C″ is assumed.

Where: ″%1$s″ is the unrecognized linkage.

Explanation: The specified linkage is unknown and extern ″C″ will be used.

User Response: Change the linkage specification.

CCN5524 The file ″%1$s″ cannot be loaded.

Where: ″%1$s″ is the name of the file.

Explanation: The file could not be loaded because of operating system errors.

User Response: Ensure that the file is correctly specified for the operating system.

CCN5525 The enum cannot be packed to the requested size of %1$s bytes.

Where: %1$s is the number of bytes specified.

Explanation: The range of values specified for the enumeration is too large to be packed into the specified number of bytes.

User Response: Change the number of bytes allowed for the enumeration or change the enumerators to have a smaller range.

CCN5526 One or more error messages have been disabled.

Explanation: An error was encountered but the error message has been suppressed.

User Response: Do not suppress the error message or fix the error.

CCN5527 The build failure may be because of an Internal Compiler Error or because a tool failed to generate a message.

Explanation: Informational message about why the build failed with no message.

User Response: Report the problem to your IBM C++ service representative.

CCN5600 The reference to ″%1$s″ is ambiguous.

Where: ″%1$s″ is the ambiguous name.

Explanation: More than one declaration was found for the reference.

User Response: Fully qualify the reference.

CCN5601 The reference to ″%1$s″ is ambiguous because ″%1$s″ is declared in base classes ″%2$s″ and ″%3$s″.

Where: ″%1$s″ is the ambiguous reference. ″%2$s″ and ″%3$s″ are two base classes.

Explanation: Multiple inheritance has supplied more than one declaration with the same name.

User Response: Fully qualify the reference or change the base classes.

CCN5602 The reference to ″%1$s″ is ambiguous because ″%1$s″ can be accessed via multiple paths to base class ″%2$s″.

Where: ″%1$s″ is the ambiguous reference. ″%2$s″ is the base class.

Explanation: Multiple inheritance has resulted in a declaration that can be reached in more than one way through the class hierarchy.

User Response: Fully qualify the reference or change the base classes.

CCN5603 The template declaration ″%1$s″ cannot be found. An extra ″template <>″ may be specified on this declaration.

Where: ″%1$s″ is the template declaration.

Explanation: Nested template explicit specializations and out-of-line declarations require a template scope for each level of nesting.

User Response: Check and correct the template scopes on the declaration.

CCN5605 This message is no longer used.

CCN5700 The previous message was produced while processing ″%1$s″.

Where: ″%1$s″ is the declaration (usually a template) that was being processed when the error occurred.

Explanation: An informational message message giving trace back information.

User Response: See the primary message.

CCN5701 The limit on nested template instantiations has been exceeded while instantiating ″%1$s″.

Where: ″%1$s″ is the last instantiation done.

Explanation: A template instantiation that requires another instantiation can set off a chain of instantiations with no end.

User Response: Change the template implementation to avoid the recursion or write an explicit specialization that will stop the instantiation chain at a reasonable point.

CCN5702 The template argument ″%1$s″ is not valid.

Where: ″%1$s″ is the template argument.

Explanation: The template argument does not match the template parameter.

User Response: Correct the template argument.

CCN5704 The definitions of ″%1$s″ and ″%2$s″ have the same linkage signature ″%3$s″.

Where: ″%1$s″ and ″%2$s″ are the two declarations. ″%3$s″ is the linkage signature.

Explanation: The two definitions have the same mangled names and the linker will be unable to distinguish them.

User Response: Remove one of the definitions or change its linkage.

CCN5705 The definition of ″%1$s″ has the same linkage signature, ″%2$s″, as a symbol from ″%3$s″.

Where: ″%1$s″ is the declaration. ″%2$s″ is the linkage signature. ″%3$s″ is the library with the conflicting symbol.

Explanation: Two definitions have the same mangled names and the linker will be unable to distinguish them.

User Response: Remove one of the definitions or change its linkage.

CCN5706 The symbol ″%1$s″ is already defined by ″%2$s″ in target ″%3$s″.

Where: ″%1$s″ is the duplicate symbol. ″%2$s″ is the source file or source library. ″%3$s″ is the target executable, library, or object file.

Explanation: A symbol is being redefined by another compilation unit.

User Response: Remove one of the symbols so that only one definition exists.

CCN5707 The symbol ″%1$s″ has the same signature as ″%2$s″ in target ″%3$s″.

Where: ″%1$s″ is the duplicate symbol. ″%2$s″ is the name of the definition that is resolving to the same symbol as ″1s″. ″%3$s″ is the target executable, library, or object file to which ″%2$s″ belongs.

Explanation: A symbol is being redefined by another compilation unit.

User Response: Remove one of the symbols so that only one definition exists.

CCN5708 The template argument %1$s does not match the corresponding template parameter of ″%2$s″.

Where: %1$s is the template argument. ″%2$s″ is the template.

Explanation: Template arguments must match the type and kind of the template parameter.

User Response: Correct the template argument.

CCN5709 The wrong number of template arguments have been specified for ″%1$s″, from line %3$s of ″%2$s″.

Where: ″%1$s″ is the template. ″%2$s″ is the source file. ″%3$s″ is the line number.

Explanation: The number of template arguments must match the number of template parameters.

User Response: Remove the extra template arguments.

CCN5710 The static function ″%1$s″ is not defined, but is referenced from ″%2$s″.

Where: ″%1$s″ is the static function, ″%2$s″ is the referencing location.

Explanation: A referenced static function must be defined.

User Response: Define the function.

CCN5711 Too few template arguments have been specified.

Explanation: The number of template arguments must match the number of template parameters.

User Response: Add the missing template arguments.

CCN5712 Too many template arguments have been specified.

Explanation: The number of template arguments must match the number of template parameters.

User Response: Remove the extra template arguments.

CCN5713 The template argument ″%1$s″ is not valid for a non-type template parameter.

Where: ″%1$s″ is the invalid argument.

Explanation: A non-type template parameter cannot be satisfied with a type.

User Response: Change the template argument to a valid value.

CCN5714 The template argument must be a type, to match the template parameter.

Explanation: Only a type-id can be used for a type template argument.

User Response: Change the template argument to a valid value.

CCN5715 The local type ″%1$s″ cannot be used in a template argument.

Where: ″%1$s″ is the local type.

Explanation: A type defined in a function body or any type compounded from a local type cannot be used as a template argument.

User Response: Change the argument to be a non-local type, or move the local type to namespace scope.

CCN5716 The template argument ″%1$s″ does not match the template parameter ″%2$s″.

Where: ″%1$s″ is the invalid argument, ″%2$s″ is the template parameter.

Explanation: A template parameter must have a template argument and a regular type template parameter cannot have a template as an argument.

User Response: Change the argument to correctly match the template parameter.

CCN5717 The template argument cannot use an unnamed type.

Explanation: An unnamed type or any type compounded from an unnamed type cannot be used as a template argument.

User Response: Change the argument to be a non-local type, or give the type a name.

CCN5718 An implicit copy assignment operator cannot be created for class with a member of type ″%1$s″.

Where: The type of the member which prohibits the generation of an implicit copy assignment operator.

Explanation: The class does not have a user specified copy assignment operator and one cannot be generated because of the type of the members of the class.

User Response: Provide a copy assignment operator.

CCN5719 The previous message was produced while processing the implicit member function ″%1$s″.

Where: The name of the member function.

Explanation: Informational message indicating which implicit member function caused the generation of the error or warning message.

User Response: See the primary message.

CCN5720 Function ″%1$s″ has internal linkage but is undefined.

Where: The name of the function that is not defined.

Explanation: A function was declared to have internal linkage, possibly because it was declared to be static, but it is not defined.

User Response: Define the function.

CCN5721 The explicit specialization ″%1$s″ must be declared before it is used.

Where: ″%1$s″ is the explicit specialization.

Explanation: A use with no explicit specialization will cause an implicit instantiation. This will conflict with the explicit specialization.

User Response: Move the use or the declaration of the explicit specialization.

CCN5722 The partial specialization ″%1$s″ must be declared before it is used.

Where: ″%1$s″ is the partial specialization.

Explanation: A use with no partial specialization will cause an implicit instantiation of the primary template. This will give different behavior than an instantiation of the partial specialization.

User Response: Move the use or the declaration of the partial specialization.

CCN5723 The inline function ″%1$s″ is referenced, but it is not defined.

Where: ″%1$s″ is the inline function.

Explanation: A referenced inline function must be defined.

User Response: Define the function.

CCN5724 The non-type template argument ″%1$s″ of type ″%2$s″ has wrapped.

Where: %1$s is the argument value and %2$s is its type.

Explanation: A non-type template argument has been provided that is outside the range for the argument type.

User Response: If this is not intended, change the argument value.

CCN5725 The physical size of an array is too large.

Explanation: The maximum allowable size for this target system has been exceeded.

User Response: Reduce the size of the array.

CCN5726 The physical size of a class or union is too large.

Explanation: The maximum allowable size for this target system has been exceeded.

User Response: Reduce the size of the class or union.

CCN5727 The static storage is too large.

Explanation: A limit on static storage has been exceeded.

User Response: Decrease the amount of storage required.

CCN5728 The keyword _Packed must be used in a typedef.

Explanation: The _Packed type specifier can only be used in a typedef declaration.

User Response: Use _Packed in a typedef declaration to declare the _Packed class type, then use the typedef name to declare the variable.

CCN5729 The keyword _Packed must be associated with a class definition.

Explanation: The _Packed specifier is only valid on a typedef declaration with a class definition.

User Response: Define the _Packed class type in the typedef declaration.

CCN5730 This message is no longer used.

CCN5731 This message is no longer used.

CCN5800 The conversion from codepage ″%1$s″ to ″%2$s″ cannot be initialized.

Where: ″%1$s″ is the source codepage. ″%2$s’ is the target codepage.

Explanation: The specified codepage does not exist.

User Response: Change the codepage specified to a valid one.

CCN5801 The character literal is empty.

Explanation: The character literal is invalid because it is empty.

User Response: Change the character literal.

CCN5802 The character literal %1$s contains more than one character.

Where: ″%1$s″ is the character literal in error.

Explanation: The character literal is invalid because it has more than one character.

User Response: Change the character literal to a single character.

CCN5803 The value of the character literal %1$s contains more bytes than sizeof(int). Only the right-most bytes are retained.

Where: ″%1$s″ is the character literal in error.

Explanation: The character literal is invalid because it has too many bytes. The extra bytes to the left are ignored.

User Response: Change the character literal.

CCN5804 The characters ″/*″ are detected in a comment.

Explanation: The start of what may be a comment has been seen inside a comment. The first string ″*/″ will finish the comment which may result in unexpected behavior if this truly is a nested comment.

User Response: Remove the nested comment or the string ″/*″ from the comment.

CCN5805 Division by zero occurs on the ″#%1$s″ directive.

Where: ″%1$s″ is the preprocessor directive in the source code.

Explanation: An attempt was made to divide by zero in a preprocessor directive.

User Response: Change the preprocessor directive to not divide by zero.

CCN5806 The parameter ″%2$s″ has already been used for the macro ″%1$s″.

Where: ″%1$s″ is the name of the preprocessor macro in error. ″%2$s″ is the reused parameter from the macro in error.

Explanation: The same identifier has been used for more than one parameter for a macro.

User Response: Change the parameter name.

CCN5807 The #elif directive has no matching #if, #ifdef, or #ifndef directive.

Explanation: The #elif directive requires a previous #if, #ifdef, or #ifndef. It may be that a #endif was added inappropriately.

User Response: Remove the #elif directive.

CCN5808 The #else directive has no matching #if, #ifdef, or #ifndef directive.

Explanation: The #else directive requires a previous #if, #ifdef, or #ifndef. It may be that a #endif was added inappropriately.

User Response: Remove the #else directive.

CCN5809 The source file is empty.

Explanation: Informational message indicating that the source file contains no preprocessing tokens.

User Response: See the primary message.

CCN5810 An empty argument is specified for parameter ″%2$s″ of the macro ″%1$s″.

Where: ″%1$s″ is the name of the macro. ″%2$2″ is the parameter receiving the empty argument.

Explanation: The argument specified to the macro is empty.

User Response: Change the argument.

CCN5811 The #endif directive has no matching #if, #ifdef, or #ifndef directive.

Explanation: The #endif directive requires a previous #if, #ifdef, or #ifndef. It may be that a #endif was added inappropriately.

User Response: Remove the #endif directive.

CCN5812 The escape sequence ″%1$s″ is out of range.

Where: ″%1$s″ is the escape sequence from the source code.

Explanation: The specified escape sequence is not valid.

User Response: Change the escape sequence.

CCN5813 One or more #endif directives are missing at the end of the file.

Explanation: There must be a #endif for every #if, #ifdef, or #ifndef. It may be that a #endif was removed inappropriately.

User Response: Add the missing #endif.

CCN5814 Expecting a macro name on the #%1$s directive but found ″%2$s″.

Where: ″%1$s″ is the preprocessor directive. ″%2$s″ is the text found where the macro name was expected.

Explanation: The text specified for the macro name is invalid.

User Response: Change the text for the macro name.

CCN5815 Expecting the end of the line on the #%1$s directive but found ″%2$s″.

Where: ″%1$s″ is the preprocessor directive. ″%2$s″ is the unexpected input.

Explanation: The end of line that was expected to terminate the preprocessing directive was not found.

User Response: Change the preprocessing directive.

CCN5816 Too many arguments are specified for the macro ″%1$s″. The extra arguments are ignored.

Where: ″%1$s″ is the name of the macro.

Explanation: The extra arguments specified for the macro are ignored.

User Response: Remove the extra arguments.

CCN5817 The comment which began on line %1$s did not end before the end of the file.

Where: ″%1$s″ is the line number on which the comment began.

Explanation: The ″*/″ ending the comment was not found before the end of the file.

User Response: Add ″*/″ to finish the comment.

CCN5818 The continuation sequence at the end of the file is ignored.

Explanation: End of file is unexpected after the continuation sequence.

User Response: Remove the continuation sequence.

CCN5819 Unable to open the file %1$s. %2$s.

Where: ″%1$s″ is the file name that could not be opened. ″%2$s″ is the text returned by the system when the file open failed.

Explanation: The file could not be opened because of the reason indicated.

User Response: Ensure that the file can be opened.

CCN5820 Unable to read the file %1$s. %2$s.

Where: ″%1$s″ is the file name that could not be opened. ″%2$s″ is the text returned by the system when the file open failed.

Explanation: The file could not be read because of the reason indicated.

User Response: Ensure that the file exists and can be read.

CCN5821 The floating point literal ″%1$s″ is out of range.

Where: ″%1$s″ is the incorrect literal.

Explanation: The floating point literal is not valid.

User Response: Change the floating point literal.

CCN5822 The name ″%1$s″ must not be defined as a macro.

Where: ″%1$s″ is the name of the reserved macro name.

Explanation: The name cannot be used as a macro.

User Response: Change the name of the macro.

CCN5823 The name ″%1$s″ must not be undefined as a macro.

Where: ″%1$s″ is the name of the reserved macro name.

Explanation: The name cannot be undefined as a macro.

User Response: Change the name of the macro.

CCN5824 The header of the #include directive is empty.

Explanation: The #include directive is improperly specified.

User Response: Change the #include specification.

CCN5825 The character ″%1$s″ is not allowed.

Where: ″%1$s″ is the character.

Explanation: The character is not valid.

User Response: Change the character.

CCN5826 The use of the ## operator in the macro ″%1$s″ is not valid.

Where: ″%1$s″ is the name of the macro in error.

Explanation: The use of the ## operator is not valid.

User Response: Change the ## operator.

CCN5827 The constant expression on the #%1$s directive contains a syntax error at ″%2$s″.

Where: ″%1$s″ is the preprocessor directive. ″%2$s″ is the token that is causing the syntax error.

Explanation: There is a syntax error in the constant expression.

User Response: Fix the syntax of the constant expression.

CCN5828 The escape sequence ″%1$s″ is not known. The backslash is ignored.

Where: ″%1$s″ is the escape sequence.

Explanation: The escape sequence is not valid and the backslash is ignored.

User Response: Remove the backslash or change the escape sequence to a valid one.

CCN5829 The suffix of the floating point literal ″%1$s″ is not valid.

Where: ″%1$s″ is the floating point literal.

Explanation: The floating point literal is improperly specified.

User Response: Change the floating point literal.

CCN5830 The suffix of the integer literal ″%1$s″ is not valid.

Where: ″%1$s″ is the floating point literal.

Explanation: The integer literal is improperly specified.

User Response: Change the integer literal.

CCN5831 The parameter list for the macro ″%1$s″ contains a syntax error at ″%2$s″.

Where: ″%1$s″ is the name of the macro. ″%2$s″ is the token that is causing the syntax error.

Explanation: There is a syntax error in the parameter list for the macro.

User Response: Fix the syntax error in the parameter list.

CCN5832 The value, ″%1$s″, of the wide character is not valid.

Where: ″%1$s″ is the value of the wide character.

Explanation: The value of the wide character is not valid.

User Response: Change the value of the wide character.

CCN5833 The multibyte character ″%1$s″ is unknown.

Where: ″%1$s″ is the multibyte character in error.

Explanation: The multibyte character is unknown.

User Response: Change the multibyte character.

CCN5834 A header name is expected on the #include directive but ″%1$s″ is found.

Where: ″%1$s″ is the unexpected text found.

Explanation: The #include directive is not valid.

User Response: Change the #include directive.

CCN5835 The file ″%1$s″ cannot be included because the maximum nesting of %2$s has been reached.

Where: ″%1$s″ is the file name. ″%2$s' is the maximum include file nesting limit for the compiler.

Explanation: The maximum number of nested include files has been reached.

User Response: Remove some of the included files or change the include structure to not nest as deeply.

CCN5836 The #include file %1$s is not found.

Where: ″%1$s″ is the file name.

Explanation: The specified include file was not found.

User Response: Ensure that the file exists, change the name of the included file, or use the include path option to specify the path to the file.

CCN5837 An incomplete argument is specified for the parameter ″%2$s″ of the macro ″%1$s″.

Where: ″%1$s″ is the name of the macro. ″%2$s″ is the macro paramter.

Explanation: The argument to the macro is invalid.

User Response: Change the argument to the macro.

CCN5838 An incomplete parameter list is specified for the macro ″%1$s″.

Where: ″%1$s″ is the acro name.

Explanation: The parameter list to the macro is incomplete.

User Response: Change the parameter list.

CCN5839 Preprocessor internal error in ″%1$s″. File ″%2$s″: Line %3$s.

Where: ″%1$s″ is the name of the compiler function at the time of the error. ″%2$s″ is the source file that was being processed at the time of the error. ″%3$s″ is the line number that was being processed at the time of the error.

Explanation: An internal error has occurred in the preprocessor.

User Response: Contact your IBM C++ service representative.

CCN5840 The integer literal ″%1$s″ is out of range.

Where: ″%1$s″ is the integer literal that is out of range.

Explanation: The integer literal is not valid.

User Response: Change the integer literal.

CCN5841 The wide character literal %1$s contains more than one character. The last character is used.

Where: ″%1$s″ is the literal.

Explanation: More than one character has been specified for a wide character literal.

User Response: Remove the extra characters from the wide character literal.

CCN5842 The line number %1$s on the #line directive must contain only decimal digits.

Where: ″%1$s″ is the invalid line number specified in the #line directive.

Explanation: The #line directive contains an invalid number.

User Response: Change the number in the #line directive.

CCN5843 Expecting a file name or the end of line on the #line directive but found ″%1$s″.

Where: ″%1$s″ is the unexpected text.

Explanation: The #line directive is invalid.

User Response: Remove the extra symbols from the #line directive.

CCN5844 Expecting a line number on the #line directive but found ″%1$s″.

Where: ″%1$s″ is the unexpected text.

Explanation: The line number specified in the #line directive is invalid.

User Response: Change the #line directive.

CCN5845 The #line value ″%1$s″ must not be zero.

Where: ″%1$s″ is the invalid value specified in the #line directive.

Explanation: The line number for a #line directive must not be zero.

User Response: Change the line number for the #line directive.

CCN5846 The #line value ″%1$s″ is outside the range 0 to 32767.

Where: ″%1$s″ is the invalid value specified in the #line directive.

Explanation: The line number for a #line directive is too large.

User Response: Change the line number for the #line directive.

CCN5847 Expected an identifier but found ″%2$s″ in the parameter list for the macro ″%1$s″.

Where: ″%1$s″ is the macro name. ″%2$s″ is the unexpected text found.

Explanation: The parameter to the macro is invalid.

User Response: Change the parameter to the macro.

CCN5848 The macro name ″%1$s″ is already defined with a different definition.

Where: ″%1$s″ is the macro name.

Explanation: An attempt is being made to redefine the macro.

User Response: Change the name of the macro being defined.

CCN5849 The octal literal ″%1$s″ contains non-octal digits.

Where: ″%1$s″ is the octal literal.

Explanation: The octal literal can only contain the digits 0-7.

User Response: Change the literal.

CCN5857 The macro name ″%1$s″ is reserved but the directive is processed.

Where: ″%1$s″ is the macro name.

Explanation: The macro name is a reserved name.

User Response: Change the name of the macro.

CCN5858 The macro name ″%1$s″ is reserved but the directive is processed.

Where: ″%1$s″ is the macro name.

Explanation: The macro name is a reserved name.

User Response: Change the name of the macro to one that is not reserved.

CCN5859 #error directive: %1$s.

Where: ″%1$s″ is the text that was specified by the #error directive in the source.

Explanation: A #error directive has been processed.

User Response: Remove the #error directive.

CCN5860 A parameter name is expected after the # operator in the macro ″%1$s″ but ″%2$s″ is found.

Where: ″%1$s″ is the macro name. ″%2$s″ is the unexpected text.

Explanation: The right operand to the # operator is invalid.

User Response: Change the right operand to the # operator.

CCN5861 Too few arguments are specified for macro ″%1$s″. Empty arguments are used.

Where: ″%1$s″ is the macro name.

Explanation: Not enough arguments have been specified for the macro.

User Response: Add more arguments to the macro.

CCN5862 The unknown preprocessing directive ″%1$s″ is ignored.

Where: ″%1$s″ is the unknown directive.

Explanation: The preprocessing directive is unknown.

User Response: Change the preprocessing directive.

CCN5863 A character literal must end before the end of the source line.

Explanation: The character literal is improperly specified.

User Response: Change the character literal.

CCN5864 A #include header must end before the end of the source line.

Explanation: The #include directive is improperly specified.

User Response: Change the #include directive.

CCN5865 A character literal must end before the end of the source line.

Explanation: The character literal is improperly specified.

User Response: Change the character literal.

CCN5866 A string literal must end before the end of the source line.

Explanation: The string literal is improperly specified.

User Response: Change the string literal.

CCN5868 A string literal must end before the end of the source line.

Explanation: The string literal is improperly specified.

User Response: Change the string literal.

CCN5869 %1$s digits are required for the universal-character-name ″%2$s″.

Where: ″%1$s″ is the required number of digits. ″%2$s″ is the universal-character-name.

Explanation: The universal-character-name is improperly specified.

User Response: Change the universal-character-name.

CCN5870 The universal-character-name ″%1$s″ is not in the allowable range for an identifier.

Where: ″%1$s″ is the universal-character name.

Explanation: The universal-character-name is improperly specified.

User Response: Change the universal-character-name.

CCN5871 Incomplete or invalid multibyte character, conversion failed.

Explanation: The multibyte character is invalid.

User Response: Change the multibyte character.

CCN5872 A string literal cannot be longer than 32765 characters.

Explanation: The string literal is too long.

User Response: Change the string literal.

CCN5900 #include search attempted to open the file ″%1$s″.

Where: ″%1$s″ is the file name.

Explanation: Informational message about the search path when attempting to find an include file.

User Response: See the primary message.

CCN5921 ″%1$s″ is defined in the file ″%2$s″ on line %3$s.

Where: ″%1$s″ is the macro name. ″%2$s″ is the file name. ″%3$s″ is the line number.

Explanation: Informational message about where a macro is defined.

User Response: See the primary message.

CCN6100 A local variable or compiler temporary variable is being used to initialize reference member ″%1$s″.

Explanation: Initializing a reference member with a temporary or local variable is dangerous since it will result in a dangling reference if the object's life-span is longer than the temporary or local variable.

User Response: Initialize the member with another object.

CCN6101 A return value of type ″%1$s″ is expected.

Explanation: The function is expected to return a value but no return statement is given.

User Response: Add a return statement to the function.

CCN6102 ″%1$s″ might be used before it is set.

Where: ″%1$s″ is the variable.

Explanation: The compiler cannot determine that the variable is initialized before it is used.

User Response: Initialize the variable.

CCN6103 The address of a local variable or temporary is used in a return expression.

Explanation: The address of a local object is being returned by the function but this object's life-span will end at the function return, resulting in a dangling reference.

User Response: Return a different value.

CCN6104 The condition evaluates to a constant value.

Explanation: The condition is a constant expression which may result in code that can never be reached or a loop that may not terminate.

User Response: Change the condition to be non-constant.

CCN6105 The condition contains a non-parenthesized assignment.

Explanation: An assignment is being performed in a condition.

User Response: Change the expression; this warning is often caused by an assignment being used when an equality comparison is desired.

CCN6106 The local type ″%1$s″ must not be used in a declaration with external linkage.

Where: ″%1$s″ is the type used in the source code declaration.

Explanation: The function has external linkage but is using a local type so the linkage signature of the function cannot be described.

User Response: Use a non-local type in the function prototype.

CCN6107 An object of abstract class ″%1$s″ cannot be created.

Where: ″%1$s″ is the class.

Explanation: The class has pure virtual functions so an object of this class type cannot be created.

User Response: Ensure that the class contains no pure virtual functions.

CCN6108 ″%1$s″ is not a valid type.

Where: ″%1$s″ is the type.

Explanation: The specific type is not a legal type.

User Response: Change the type.

CCN6109 The use of undefined class ″%1$s″ is not valid.

Where: ″%1$s″ is the class.

Explanation: The use requires that the type be defined and not just declared.

User Response: Define the class.

CCN6110 The referenced type ″%1$s″ contains a circular reference back to ″%2$s″.

Where: ″%1$s″ and ″%2$s″ are the types in question.

Explanation: The two types contain references to each other that both require definitions.

User Response: Change the first class to only require a declaration of the second class.

CCN6111 Only function declarations can have default arguments.

Explanation: An attempt has been made to have default arguments for a parameter in a declaration that is not a function declaration.

User Response: Remove the default initializers.

CCN6112 ″%1$s″ is a pure virtual function.

Where: ″%1$s″ is the name of the function.

Explanation: Informational message for listing pure virtual functions.

User Response: See the primary message.

CCN6113 The class template name ″%1$s″ must be followed by a < in this context.

Where: ″%1$s″ is the name of the template class.

Explanation: The template must have its template arguments specified.

User Response: Add the < and the appropriate template arguments followed by >.

CCN6114 ″%1$s″ is not allowed as a function return type.

Where: ″%1$s″ is the type that the function is attempting to return.

Explanation: The return type of the function is not valid.

User Response: Change the function return type.

CCN6115 ″%1$s″ cannot be declared to have type ″void″.

Where: ″%1$s″ is the name of the declaration.

Explanation: The type ″void″ is not valid for this declaration.

User Response: Change the type.

CCN6116 If ″%1$s″ is a function name, one of its parameters may contain an undeclared type name.

Where: ″%1$s″ is the name of the attempted function or variable declaration.

Explanation: A function declaration that has an unknown type as a parameter may have been incorrectly parsed as a variable declaration with a paren-style initializer.

User Response: See the primary message.

CCN6117 ″%1$s″ cannot use the abstract class ″%2$s″ as the type of an object, parameter type, or return type.

Where: ″%1$s″ is what is attempting to use the abstract base class ″%2$s″.

Explanation: The class has pure virtual functions so an object cannot be created.

User Response: Change the type of the object being created.

CCN6118 The declaration of ″%1$s″ uses the undefined class ″%2$s″ when the class must be complete.

Where: ″%1$s″ is the name of the declaration. ″%2$s″ is the type being declared.

Explanation: The usage requires the class to be defined.

User Response: Define the class.

CCN6120 ″using %1$s″ must refer to a member of a base class.

Where: ″%1$s″ is the argument of the using directive.

Explanation: The using declaration must refer to a member of a base class.

User Response: Change the declaration.

CCN6121 ″%1$s″ is a class member and can be declared only in a member declaration.

Where: ″%1$s″ is a class member.

Explanation: A using declaration for a class member shall be a member declaration

User Response: Remove the using declaration, or move it into a class derived from the class that contains the member declaration.

CCN6122 A non-type template parameter cannot have type ″%1$s″.

Where: ″%1$s″ is the invalid type.

Explanation: Only integral, enumeration, pointer or reference types (or cv-qualified versions) are allowed as non-type template parameters.

User Response: Correct the non-type template parameter.

CCN6123 An initializer is not allowed for ″%1$s″.

Where: ″%1$s″ is the name of the declaration.

Explanation: An initializer has been specified for a declaration that does not create an object.

User Response: Remove the initializer.

CCN6124 A union cannot contain a static data member.

Explanation: Static data members have external linkage. They cannot be used in unions, because members of a union share the same memory.

User Response: Change the union into a class or struct, or remove the static data member.

CCN6125 The data member ″%1$s″ cannot have the same name as its containing class.

Where: ″%1$s″ is the name of a class data member.

Explanation: Every data member of a class must have a name different from the name of the containing class

User Response: Change the name of the data member so that it is not the same as the class name.

CCN6126 The static data member ″%1$s″ is not allowed in a local class.

Where: ″%1$s″ is a data member of a local class.

Explanation: Since static data members have external linkage it makes no sense to have one inside a local class. If this were permitted, the static data member would be visible in scopes where the class itself is not visible.

User Response: Remove the static data member or move the class to global scope.

CCN6127 Only static data members with const integral or const enumeration type can specify an initializer in the class definition.

Explanation: The declaration of a static data member is not a definition. The definition should appear in a namespace scope enclosing the class that contains this member. Only static data members of const integral or const enumeration type may be initialized inside the class declaration. In this case, they must still be defined in the enclosing scope without an initializer.

User Response: Move the initializer to the definition in the containing scope, or make the type a const integral or const enumeration.

CCN6128 The bit-field ″%1$s″ must have integral or enumeration type.

Where: ″%1$s″ is the name of the bit-field.

Explanation: A bit-field is used to represent a sequence of bits. Only integral or enumeration types makes sense for bit-fields.

User Response: Change the type of the bit-field or remove the bit-field.

CCN6129 The ″mutable″ specifier must not be applied to a member with type ″%1$s″.

Where: ″%1$s″ is the type of the data member.

Explanation: The mutable specifier cannot be applied to const, static or reference members.

User Response: Remove the mutable specifier from the data member or change the type of the data member

CCN6130 A static data member cannot be a direct or indirect member of an unnamed class.

Explanation: Static data members are defined and accessed using the name of the class in which they are defined. If the class has no name, the static data member cannot be defined or accessed.

User Response: Give the class a name, or make the data member non-static.

CCN6131 A zero-length bit-field must not have a name.

Explanation: Bit-fields with zero-length are used to specify alignment of the next bit-field at the boundary of an allocation unit. They have no data and are therefore not accessed for any reason.

User Response: Change the length of the bit-field or remove the name.

CCN6132 ″%1$s″ must not be a member of a union. ″%2$s″ has a non-trivial copy assignment operator.

Explanation: Unions can only contain members that do not have copy assignment operators.

User Response: Change the member to be a POD-type.

CCN6133 A union must not contain a member of type ″%1$s″.

Where: ″%1$s″ is the type.

Explanation: Reference variables are not allowed in unions.

User Response: Change the type of the member.

CCN6134 An anonymous %1$s must not have private or protected members.

Where: %1$s is the keyword union, struct or class.

Explanation: Only public members are allowed in anonymous aggregates.

User Response: Ensure that all members are public.

CCN6135 The anonymous %1$s member ″%2$s″ must not have the same name as its containing class.

Where: ″%1$s″ is either union, struct or class. ″%2$s″ is the name of the member.

Explanation: Every data member of a class must have a name different from the name of the containing class. Members of anonymous struct, class, or union are referenced as members of their containing class, so their name must also be different from the name of containing class.

User Response: Change the name of the member.

CCN6136 ″%1$s″ cannot be a union member, because ″%2$s″ has a non-trivial constructor.

Where: ″%1$s″ is the declaration of the union member ″%2$s″ is the name of the class that has a non-trivial constructor.

Explanation: A trivial constructor is created by the compiler for a class with: no virtual functions and no virtual base classes. All the direct base classes of its class must have trivial constructors, and all of its nonstatic data members that are of class type have must have trivial constructors. An object with a non-trivial constructor may not be a member of a union.

User Response: Change the union to a struct or a class or remove the member which has a non-trivial constructor.

CCN6137 ″%1$s″ cannot be a union member, because ″%2$s″ has a non-trivial destructor.

Where: ″%1$s″ is the declaration of the union member. ″%2$s″ is the name of the class that has a non-trivial destructor.

Explanation: Unions can only contain members that do not have destructors.

User Response: Change the member to be a POD-type.

CCN6138 Ellipsis (...) cannot be used for ″%1$s″.

Where: ″%1$s″ is the function.

Explanation: An overloaded operator cannot have an ellipsis as a parameter.

User Response: Change the ellipsis parameter.

CCN6139 An exception-specification can appear only in a function or pointer declaration.

Explanation: An exception-specification is not valid for this type.

User Response: Remove the exception-specification.

CCN6140 The member ″%1$s″ must be declared in its containing class definition.

Where: ″%1$s″ is the member.

Explanation: The member that is being defined out of line is not declared in the class.

User Response: Declare the variable or function as a member of the class.

CCN6141 An anonymous %1$s can define only non-static data members.

Where: ″%1$s″ is the keyword union, struct, or class.

Explanation: Static members are not allowed in anonymous aggregates.

User Response: Remove the static member declaration.

CCN6142 ″%1$s″ is ill-formed because ″%2$s″ does not have a unique final overrider.

Where: ″%1$s″ is the name of the derived class. ″%2$s″ is the qualified name of the virtual function with no final overrider.

Explanation: The virtual function has more than one final overrider because of virtual base classes.

User Response: Ensure that only base class has a final overrider for the function or define the virtual function in the class.

CCN6143 ″%1$s″ cannot be used as a base class because it contains a zero-dimension array.

Where: ″%1$s″ is the base class.

Explanation: The base class cannot be used since it contains an array that has zero elements.

User Response: Change the base class.

CCN6144 All array dimensions for non-static members must be specified and be greater than zero.

Explanation: An array dimension is missing or is negative.

User Response: Ensure that all dimensions are specified as non-negative numbers.

CCN6145 A using-directive cannot appear in a class scope.

Explanation: Using directives can only be specified in namespace or lexical block scope.

User Response: Remove the using directive.

CCN6146 The enumerator ″%1$s″ cannot have the same name as its containing class.

Where: ″%1$s″ is the enumerator.

Explanation: This is a name collision.

User Response: Change the name of either the enumerator or the class.

CCN6147 ″%1$s″ cannot be declared as inline or static.

Where: ″%1$s″ is the function name.

Explanation: There are restrictions on ″main″ since it is the program starting point.

User Response: Remove the inline or static specifiers.

CCN6148 The non-member function ″%1$s″ cannot be declared ″%2$s″.

Where: ″%1$s″ is the name of the function. ″%2$s″ is the specifier.

Explanation: The specifier is only valid for member functions.

User Response: Remove the specifier.

CCN6149 ″%1$s″ is not originally declared in namespace ″%2$s″.

Where: ″%1$s″ is the declared name. ″%2$s″ is the namespace.

Explanation: The qualifiers specify a namespace which does not have a corresponding declaration.

User Response: Change the qualifiers to refer to the proper namespace.

CCN6150 A constructor for ″%1$s″ cannot be declared ″%2$s″.

Where: ″%1$s″ is the struct, or class. ″%2$s″ is the specifier.

Explanation: The specifier is not valid for a constructor.

User Response: Remove the specifier.

CCN6151 When the first parameter to the constructor has type ″%1$s″, the constructor must have other parameters without default arguments.

Where: ″%1$s″ is the type.

Explanation: This is an ill-formed copy constructor since the first parameter is not a reference.

User Response: Change the first parameter to be a reference to make this a copy constructor.

CCN6152 The destructor for ″%1$s″ cannot be declared ″%2$s″.

Where: ″%1$s″ is the struct, or class. ″%2$s″ is the specifier.

Explanation: The specifier is not valid for a destructor.

User Response: Remove the specifier.

CCN6153 A destructor must not have a return type or parameter.

Explanation: A return type or parameter has been specified for a destructor.

User Response: Remove the return type or parameter.

CCN6154 The destructor ″%1$s″ must not be declared as a template.

Where: ″%1$s″ is the destructor.

Explanation: A destructor must not be a member template.

User Response: Remove or change the destructor to be a regular non-template destructor.

CCN6155 The static member function ″%1$s″ must not be declared ″%2$s″.

Where: ″%1$s″ is the name of the function. ″%2$s″ is the specifier.

Explanation: A static function cannot have cv-qualifiers.

User Response: Remove the cv-qualifiers.

CCN6156 A conversion operator must not have parameters.

Explanation: A conversion operator has been specified with parameters.

User Response: Remove the parameters.

CCN6157 A conversion operator must not have type ″%1$s″.

Where: ″%1$s″ is the type.

Explanation: A conversion operator has been specified with void type.

User Response: Remove the void specifier.

CCN6158 The function template ″%1$s″ must not be declared as virtual.

Where: ″%1$s″ is the function.

Explanation: A member function template cannot be virtual.

User Response: Change the function so that it is not virtual or not a template.

CCN6159 The ″%1$s″ qualifier must not be applied to ″%2$s″.

Where: ″%1$s″ is the qualifier. ″%2$s″ is the declarator.

Explanation: The qualifier is not valid for this declaration.

User Response: Remove the qualifier.

CCN6160 The virtual function ″%1$s″ is not allowed in a union.

Where: ″%1$s″ is the name of the function.

Explanation: Unions cannot have virtual member functions.

User Response: Remove the virtual specifier.

CCN6161 The default arguments for ″%1$s″ must not be followed by uninitialized parameters.

Where: ″%1$s″ is the name of the function.

Explanation: All parameters following a parameter with a default initializer must also have default initializers.

User Response: Add default initializers for all parameters after the first parameter with a default initializer.

CCN6162 The pure-specifier (= 0) is not valid for the non-virtual function ″%1$s″.

Where: ″%1$s″ is the name of the function.

Explanation: The pure-specifier (= 0) is used to state that a virtual function does not have a definition. It has no meaning for non-virtual functions.

User Response: Make the function virtual or remove the pure-specifier.

CCN6163 The exception-specification for ″%1$s″ is less restrictive than the exception-specification for ″%2$s″.

Where: ″%1$s″ is the overriding function. ″%2$s″ is the original function.

Explanation: The exception specification for an overriding function must not list more types than the exception specification for the original function.

User Response: Match the exception specification for the overriding function with the original function or modify the exception specification of the original function.

CCN6164 The return type for ″%1$s″ differs from the return type of ″%2$s″ that it overrides.

Where: ″%1$s″ and ″%2$s″ are the names of the functions.

Explanation: When overriding a function, the name, parameters and the return type should match.

User Response: Modify the return type of the overriding function to match the original function.

CCN6165 The virtual function ″%1$s″ is not a valid override of ″%2$s″ because the qualifiers are not compatible.

Where: ″%1$s″ is the function. ″%2$s″ is the function being overridden.

Explanation: The return for an override must be more cv-qualified than the function in the base class.

User Response: Add the missing qualifiers to the override.

CCN6166 The virtual function ″%1$s″ is not a valid override because ″%2$s″ is an inaccessible base class of ″%3$s″.

Where: ″%1$s″ is the function. ″%2$s″ is the base class. ″%3$s″ is the derived class.

Explanation: The override is not correct because the base class containing the function is not accessible.

User Response: Remove the override.

CCN6167 The virtual function ″%1$s″ is not a valid override because ″%2$s″ is an ambiguous base class of ″%3$s″.

Where: ″%1$s″ is the function. ″%2$s″ is the base class. ″%3$s″ is the derived class.

Explanation: The override is not correct because there are multiple base classes containing the function.

User Response: Remove the override.

CCN6168 The virtual function ″%1$s″ is not a valid override because ″%2$s″ is not a base class of ″%3$s″.

Where: ″%1$s″ is the function. ″%2$s″ is the base class. ″%3$s″ is the derived class.

Explanation: The override is not correct because the return type is not complete nor the containing class.

User Response: Change the return type to be a complete class or the containing class.

CCN6169 The function template ″%1$s″ cannot have default template arguments.

Where: ″%1$s″ is the function template.

Explanation: Default template arguments are not allowed on a function template.

User Response: Remove the default template arguments.

CCN6170 Both ″main″ and ″WinMain″ are defined.

Explanation: Only one of ″main″ and ″WinMain″ can be defined in a program.

User Response: Remove either ″main″ or ″WinMain″.

CCN6171 The friend function ″%1$s″ cannot be defined in a local class.

Where: ″%1$s″ is the friend function.

Explanation: A class defined in a function body can contain a definition of a friend function.

User Response: Remove the definition of the friend in the local class.

CCN6172 More than one function ″%1$s″ has non-C++ linkage.

Where: ″%1$s″ is the function.

Explanation: Only functions with C++ linkage can be overloaded.

User Response: Change the name of the function so that it is unique.

CCN6173 ″%1$s″ is not a valid parameter type.

Where: ″%1$s″ is the type.

Explanation: The type of the parameter is not valid.

User Response: Change the type of the parameter.

CCN6174 The member ″%1$s″ is not declared as a template in its containing class definition.

Where: ″%1$s″ is the member.

Explanation: This out-of-line template class member does not exist in the class template.

User Response: Declare the member in the class template or remove the out-of-line declaration.

CCN6175 The class template partial specialization ″%1$s″ does not match the primary template ″%2$s″.

Where: ″%1$s″ is the partial specialization, ″%2$s″ is the primary template.

Explanation: Either both the primary template and the partial specialization must be unions or neither of them must be unions.

User Response: Make the class key match.

CCN6176 ″%1$s″ is declared with a conflicting linkage.

Where: ″%1$s″ is the declarator.

Explanation: The linkage is not compatible with the linkage specified in a previous declaration.

User Response: Change the linkage of one of the declarations so that they are compatible.

CCN6177 Only variables with static storage can be declared to have thread local storage.

Explanation: The __thread is specified but the declaration is not for a variable, or the variable is not declared static.

User Response: Remove the __thread specifier.

CCN6178 ″%1$s″ is declared to have both %2$s and %3$s linkage.

Where: ″%1$s″ is the declarator. ″%2$s″ is the linkage specifier. ″%3$s″ is the linkage specifier.

Explanation: The linkage is not compatible with the linkage specified in a previous declaration.

User Response: Change the linkage of one of the declarations so that they are compatible.

CCN6179 ″%1$s″ contains conflicting linkages.

Where: ″%1$s″ is the declaration.

Explanation: The linkage is not compatible with the linkage specified in a previous declaration.

User Response: Change the linkage of one of the declarations so that they are compatible.

CCN6180 Namespace ″%1$s″ must be global.

Where: ″%1$s″ is the namespace.

Explanation: A namespace can only be declared within another namespace or in the global namespace.

User Response: Move the namespace to be within another namespace.

CCN6181 The number of function parameters exceeds the target operating system limit of %1$s.

Where: %1$s is the maximum number of function parameters allowed.

Explanation: Too many function parameters have been specified.

User Response: Reduce the number of function parameters.

CCN6182 ″%1$s″ must have two or more parameters.

Where: ″%1$s″ is the function.

Explanation: The declaration of operator new does not have enough parameters.

User Response: Ensure that the function has at least two parameters.

CCN6183 The non-member function ″%1$s″ must have at least one parameter of type class or enumeration, or a reference to class or enumeration.

Where: ″%1$s″ is the function.

Explanation: The operator overload does not have the correct type for its parameters.

User Response: Change the types of the parameters.

CCN6184 Wrong number of parameters for ″%1$s″.

Where: ″%1$s″ is the function.

Explanation: The declaration for the operator overload does not have the correct number of parameters.

User Response: Change the declaration to have the proper number of parameters.

CCN6185 ″%1$s″ must be a non-static member function.

Where: ″%1$s″ is the function.

Explanation: The operator overload is only valid as a non-static member function.

User Response: Change the declaration to be a non-static member function.

CCN6186 The last parameter for postfix ″%1$s″ must be of type ″int″.

Where: ″%1$s″ is the function.

Explanation: The last parameter for the operator overload must be of type int.

User Response: Change the last parameter to be of type int.

CCN6187 ″%1$s″ must not have default arguments.

Where: ″%1$s″ is the function.

Explanation: The overloaded operator must not have default arguments.

User Response: Remove the default arguments.

CCN6188 The return type for the ″%1$s″ must not be the containing class.

Where: ″%1$s″ is the operator.

Explanation: The return type for the overloaded function cannot be the containing class.

User Response: Change the return type.

CCN6189 The return type for ″operator new″ must be ″void *″.

Explanation: The specified return type is invalid.

User Response: Change the return type.

CCN6190 The first parameter for ″operator new″ must have type ″size_t″.

Explanation: The type of the first parameter is incorrect.

User Response: Change the type of the first parameter.

CCN6191 The first parameter of ″operator new″ cannot have a default argument.

Explanation: It is invalid to specify a default argument for ″operator new″.

User Response: Remove the default argument.

CCN6192 ″%1$s″ must not be declared static in global scope.

Where: ″%1$s″ is the function.

Explanation: Overloaded versions of ″operator new″ and ″operator delete″ must not be declared static.

User Response: Remove the static specifier.

CCN6193 The member function ″%1$s″ must not be declared virtual.

Where: ″%1$s″ is the member function.

Explanation: ″Operator new″ and ″operator delete″ cannot be declared virtual in a member list.

User Response: Remove the virtual specifier.

CCN6194 ″%1$s″ must be a class member function or a global function.

Where: ″%1$s″ is the function.

Explanation: The scope for the overloaded ″operator new″ or ″operator delete″ is invalid.

User Response: Remove the declaration.

CCN6195 The return type for ″operator delete″ must be ″void″.

Explanation: A return type other than ″void″ has been specified for ″operator delete″.

User Response: Change the return type to be ″void″.

CCN6196 The return type cannot be ″%1$s″ because ″%2$s″ does not have an ″operator->″ function.

Where: ″%1$s″ is the type. ″%2$s″ is the class or struct.

Explanation: The return type must have an ″operator->″ function.

User Response: Add an ″operator->″ function to the return type.

CCN6197 Parameter number %1$s for ″operator delete″ must have type ″%2$s″.

Where: ″%1$s″ is the function parameter. ″%2$s″ is the type.

Explanation: The parameter has the wrong type.

User Response: Change the type of the parameter.

CCN6198 Too many parameters are specified for ″operator delete″.

Explanation: There are too many parameters specified.

User Response: Remove the extra parameters.

CCN6199 ″main″ must have a return type of type ″int″.

Explanation: A return type other than ″int″ has been specified for ″main″.

User Response: Change the return type of ″int″ to be ″int″.

CCN6200 An ellipsis (...) handler must not be followed by another handler.

Explanation: An ellipsis handler will match all thrown objects, and the handlers are tried in the order that they are specified. Therefore the ellipsis handler must be last.

User Response: Move the ellipsis handler to be the last handler.

CCN6201 A ″new″ expression with type ″%1$s″ must have an initializer.

Where: ″%1$s″ is the type.

Explanation: A const type must be initialized even when it is allocated with new.

User Response: Add an initializer.

CCN6202 No candidate is better than ″%1$s″.

Where: ″%1$s″ is the match.

Explanation: Informational message indicating one of the best matches for operator overloading.

User Response: See the primary message.

CCN6203 The conversion from ″%1$s″ to ″%2$s″ matches more than one conversion function.

Where: ″%1$s″ and ″%2$s″ are the types.

Explanation: There is more than one conversion sequence so it is an ambiguous conversion.

User Response: Provide a closer matching conversion.

CCN6204 The conversion matches ″%1$s″.

Where: ″%1$s″ is the conversion sequence.

Explanation: Informational message indicating a matched conversion sequence.

User Response: See the primary message.

CCN6205 The error occurred while converting to parameter %1$s of ″%2$s″.

Where: ″%1$s″ is the parameter number. ″%2$s″ is the function.

Explanation: Informational message about conversion sequences.

User Response: See the primary message.

CCN6206 The class template instantiation of ″%1$s″ is ambiguous.

Where: ″%1$s″ is the template.

Explanation: The instantiation cannot be performed since the template is not uniquely identified.

User Response: Qualify the instantiation to make it uniquely identify a template.

CCN6207 The template arguments match ″%1$s″.

Where: ″%1$s″ is the matched template.

Explanation: Informational message indicating what the template arguments match.

User Response: See the primary message.

CCN6208 The use of ″%1$s″ is not valid.

Where: ″%1$s″ is the invalid name.

Explanation: The name is being incorrectly used.

User Response: Fix the usage of the name.

CCN6209 The name lookup in the context of ″%1$s″ resolved to ″%2$s″.

Where: ″%1$s″ is the context. ″%2$s″ is the resolution.

Explanation: Informational message indicating the resolution of the name.

User Response: See the primary message.

CCN6210 Name lookup in the context of the expression resolved to ″%1$s″.

Where: ″%1$s″ is the resolution.

Explanation: Informational message indicating what the resolution of the name.

User Response: See the primary message.

CCN6211 The conversion type must represent the same type in the context of the expression as in the context of the class of the object expression.

Explanation: The conversion type is resolved in the left side of the member access and in the current scope and it can only resolve in one or it must resolve to the same entity in both.

User Response: Change the context so that the lookups match.

CCN6212 The type of the conversion function cannot be resolved.

Explanation: Some names in the type of the conversion function are not declared.

User Response: Change the conversion function so that all elements are declared.

CCN6213 The temporary for the throw expression is of type ″%2$s″ and cannot be initialized with an expression of type ″%1$s″.

Where: ″%2$s″ is the type of the throw expression. ″%1$s″ is the initialization type.

Explanation: Throw expressions throw a copy (rather than the object itself) and the temporary cannot be initialized with the given expression.

User Response: Change the initializer or provide appropriate constructors.

CCN6214 The member expression resolves to the type ″%1$s″.

Where: ″%1$s″ is the type being accessed.

Explanation: The left side of the class member access refers to type ″%1$s″.

User Response: Change the class member access expression.

CCN6215 ″%1$s″ must not have an initializer list.

Where: ″%1$s″ is the function.

Explanation: Only constructors can have constructor initializer lists and this function is not a constructor.

User Response: Remove the constructor initializer list.

CCN6216 The unqualified member ″%1$s″ must be qualified with ″%2$s::″ and preceded by an ″&″ to form an expression with type pointer-to-member.

Where: ″%1$s″ is the member. ″%2$s″ are the qualifiers.

Explanation: A pointer-to-member expression is of the form: ″&className::member ″.

User Response: Add the qualifiers and address operator.

CCN6217 The second and third operands of the conditional operators must not both be throw expressions.

Explanation: Only one of the second and third operands in a ternary operator can be a throw expression.

User Response: Change one of the second and third operators to not be a throw expression or replace the ternary expression with a conditional statement.

CCN6218 When defining the implicitly declared function ″%1$s″, the header ″<new>″ should be included.

Where: ″%1$s″ is the function being implicitly declared.

Explanation: The header ″<new>″ contains declarations that are necessary for creating some implicitly declared functions and must therefore be included using the #include directive.

User Response: Include the header ″<new>″ using an include directive.

CCN6219 ″%1$s″ must be preceded by an ″&″ to form an expression with type pointer-to-member.

Where: ″%1$s″ is the member.

Explanation: A non-static member must be associated with an object.

User Response: Add the address operator.

CCN6220 The two type-names used in the explicit destructor call, %1$s::˜%1$s, must refer to the same type.

Where: ″%1$s″ is the expected destructor specification.

Explanation: The form used to indicate a destructor in a pseudo-destructor call is not valid.

User Response: Change the specification of the destructor.

CCN6221 The explicit destructor call must be invoked for an object.

Explanation: An attempt is being made to call a destructor without an object.

User Response: Call the destructor as a member access on an object.

CCN6222 The destructor type ″%1$s″ does not match the object type ″%2$s″.

Where: ″%1$s″ is the type of the destructor. ″%2$s″ is the type of the object.

Explanation: The destructor indicated does not match the type of the object.

User Response: Change the destructor to match the type of the object.

CCN6223 ″%1$s″ is not valid as an identifier expression.

Where: ″%1$s″ is the invalid form for an identifier.

Explanation: The form of the identifier is invalid.

User Response: Change the form to a valid form for an identifier.

CCN6224 ″%1$s″ cannot be dynamically cast to ″%2$s″ because ″%1$s″ does not declare or inherit virtual functions.

Where: ″%1$s″ is the source class. ″%2$s″ is the target class.

Explanation: Only polymorphic classes can be dynamically cast.

User Response: Remove the dynamic cast.

CCN6225 Name lookup did not find ″%1$s″ in the context of the template definition.

Where: ″%1$s″ is the unresolved name.

Explanation: This may cause an error when the template is instantiated. Declarations for non-dependent names are resolved in the template definition.

User Response: Correct the unresolved name by removing the reference or declaring it.

CCN6226 Declarations for non-dependent names are resolved in the template definition.

Explanation: This is a submessage.

User Response: See the primary message.

CCN6227 ″%1$s″ does not depend on a template argument.

Where: ″%1$s″ is the name that is not dependent on the template.

Explanation: This is a submessage.

User Response: See the primary message.

CCN6228 Argument number %1$s is an lvalue of type ″%2$s″.

Where: ″%1$s″ is the argument number. ″%2$s″ is the lvalue type.

Explanation: Informational message describing the type of a parameter to a function.

User Response: See the primary message.

CCN6229 Argument number %1$s is an rvalue of type ″%2$s″.

Where: ″%1$s″ is the argument number. ″%2$s″ is the rvalue type.

Explanation: Informational message describing the type of a parameter to a function.

User Response: See the primary message.

CCN6230 Argument number 1 is the implicit ″this″ argument.

Explanation: Informational message describing the implicit ″this″ argument in a member function.

User Response: See the primary message.

CCN6231 The conversion from argument number %1$s to ″%2$s″ uses %3$s.

Where: %1$s is the argument number. ″%2$s″ is the parameter type. %3$s is more detailed text.

Explanation: Informational message describing a conversion sequence.

User Response: See the primary message.

CCN6232 ″%1$s″

Where: ″%1$s″ is more detailed generated text.

Explanation: Informational message describing a standard conversion sequence.

User Response: See the primary message.

CCN6233 ″%1$s″ followed by ″%2$s″

Where: More detailed generated text.

Explanation: Informational message describing a standard conversion sequence.

User Response: See the primary message.

CCN6234 ″%1$s″ followed by ″%2$s″ followed by ″%3$s″

Where: More detailed generated text.

Explanation: Informational message describing a standard conversion sequence.

User Response: See the primary message.

CCN6235 the user-defined conversion ″%1$s″

Where: ″%1$s″ is the name of a user-defined conversion function.

Explanation: Informational message describing a user-defined conversion sequence.

User Response: See the primary message.

CCN6236 the user-defined conversion ″%1$s″ followed by ″%2$s″

Where: ″%1$s″ is the name of a user-defined conversion function. ″%2$s″ is more detailed generated text.

Explanation: Informational message describing a user-defined conversion sequence.

User Response: See the primary message.

CCN6237 %1$s followed by the user-defined conversion ″%2$s″

Where: %1$s is more detailed generated text. ″%2$s″ is the name of a user-defined conversion function.

Explanation: Informational message describing a user-defined conversion sequence.

User Response: See the primary message.

CCN6238 ″%1$s″ followed by the user-defined conversion ″%2$s″ followed by %3$s

Where: ″%1$s″ is more detailed generated text. ″%2$s″ is the name of a user-defined conversion function. ″%3$s″ is more detailed generated text.

Explanation: Informational message describing a user-defined conversion sequence.

User Response: See the primary message.

CCN6239 an ellipsis conversion sequence

Explanation: Informational message about a conversion sequence.

User Response: See the primary message.

CCN6240 the resolved overloaded function ″%1$s″

Where: ″%1$s″ is the function.

Explanation: Informational message about a conversion sequence.

User Response: See the primary message.

CCN6260 An object of type ″%2$s″ cannot be constructed from an rvalue of type ″%1$s″.

Where: ″%2$s″ is the type being constructed. ″%1$s″ is the type of the expression.

Explanation: There is no valid way to construct the desired object from the given type.

User Response: Change the expression.

CCN6261 The qualified member ″%1$s″ should not be in parentheses when forming an expression with type pointer-to-member.

Where: ″%1$s″ is the member.

Explanation: Informational message indicating that removing the parentheses may resolve the error.

User Response: See the primary message.

CCN6262 The scope of ″%1$s″ extends only to the end of the for-statement.

Where: ″%1$s″ is the variable.

Explanation: Informational message indicating the scoping of variables introduced in for-statements. This behavior is different in the language standard than in previous levels of the working draft.

User Response: Move the declaration above the for-statement.

CCN6263 Build with ″lang(ISOForStatementScopes, no)″ to extend the scope of the for-init-statement declaration.

Explanation: Informational message describing a compatibility option.

User Response: See the primary message.

CCN6264 The template argument must be preceded by an ampersand (&).

Explanation: The template argument is expected to be the address of an object.

User Response: Add the address operator.

CCN6265 The template argument must be the address of an object or function with extern linkage.

Explanation: For example string literals are not allowed because they have internal linkage.

User Response: Correct the template argument.

CCN6266 A template argument with type ″%1$s″ cannot be converted to a template parameter with type ″%2$s″.

Where: ″%1$s″ is the argument type. ″%2$s″ is the parameter type.

Explanation: Only certain standard conversion sequences can be applied.

User Response: Correct the template argument type.

CCN6267 ″%1$s″ is declared with internal linkage in source ″%2$s″.

Where: ″%1$s″ is the variable. ″%2$s″ is the source.

Explanation: Informational message about where an object is declared with internal linkage.

User Response: See the primary message.

CCN6268 ″%1$s″ conflicts with the definition in source ″%2$s″ because ″%3$s″ has internal linkage.

Where: ″%1$s″ is the variable or function. ″%2$s″ is the source. ″%3$s″ is the other variable or function with internal linkage.

Explanation: The variable or function is defined as static in another source file.

User Response: Remove the static from the other definition.

CCN6269 The template argument for the non-type template parameter of type ″%1$s″ must be an integral constant expression.

Where: ″%1$s″ it the template parameter type.

Explanation: Only constant expressions are allowed for integral or enumeration non-type template arguments.

User Response: Correct the non-type template parameter.

CCN6270 A function or object name must be expressed as an id-expression.

Explanation: A function or object name used as a non-type template argument must be an id-expression with external linkage.

User Response: Correct the template argument to be a name with external linkage.

CCN6271 The ″sizeof″ operator cannot be applied to a bit-field.

Explanation: It is invalid to use the ″sizeof″ operator on a bit-field.

User Response: Remove the ″sizeof″ operator.

CCN6272 The incomplete class ″%1$s″ is not a valid ″catch″ type.

Where: ″%1$s″ is the class.

Explanation: Only complete types can be used in the type for catch handlers but the specified type has only been declared and not defined.

User Response: Define the type.

CCN6273 A pointer or reference to the incomplete class ″%1$s″ is not a valid ″catch″ type.

Where: ″%1$s″ is the incomplete class type.

Explanation: Only pointers to complete types can be used in the type for catch handlers but the type has only been declared and not defined.

User Response: Change the type in the catch or define the class.

CCN6274 The ″catch(%1$s)″ cannot be reached because of a previous ″catch(%2$s)″.

Where: ″%1$s″ is the current handler. ″%2$s″ is the previous handler.

Explanation: Catch handlers are tried sequentially and this catch is unreachable because a previous handler catches everything that this handler can catch.

User Response: Remove or change the handler.

CCN6275 Too many explicit template arguments are specified for ″%1$s″.

Where: ″%1$s″ is the template.

Explanation: The number and type of template arguments must match the template parameters.

User Response: Remove the extra template arguments.

CCN6276 The explicit template specialization ″%1$s″ matches more than one template.

Where: ″%1$s″ is the explicit specialization.

Explanation: The explicit specialization of this function matches multiple function templates. Probably because of allowable non-type template argument conversions.

User Response: Remove the explicit specialization, remove one of the primary templates, or add namespaces to separate the templates.

CCN6277 The explicit template specialization ″%1$s″ does not match any template.

Where: ″%1$s″ is the explicit specialization.

Explanation: An explicit specialization mus specialize a primary template.

User Response: Declare the primary template or correct the explicit specialization.

CCN6278 The deduced type ″%1$s″ does not match the specialized type ″%2$s″.

Where: ″%1$s″ is the deduced type, ″%2$s″ is the specialized type.

Explanation: The template argument type deduced from the function call does not match the type in the specialization.

User Response: Explicitly specify the template arguments or change the call.

CCN6279 A return statement cannot appear in a handler of the function-try-block of a constructor.

Explanation: A return statement is in a handler for a function-try-block of a constructor.

User Response: Remove the return statement.

CCN6280 An rvalue of type ″%1$s″ cannot be converted to ″%2$s″.

Where: ″%1$s″ is the original type. ″%2$s″ is the target type.

Explanation: No conversion sequence exists for converting ″%1$s″ to ″%2$s″.

User Response: Change the types or provide conversion functions.

CCN6281 ″offsetof″ cannot be applied to ″%1$s″. It is not a POD (plain old data) type.

Where: ″%1$s″ is the type.

Explanation: ″offsetof″ cannot be applied to a class that is not a POD. POD types do not have non-static pointers-to-member, non-POD members, destructors nor copy assignment operators (ie, they are similar to C-style structs).

User Response: Change the type to be a POD type.

CCN6282 An enumerator from an enumeration that is in error is being referenced.

Explanation: This is a cascading error caused by an error in the definition of the enumeration.

User Response: Fix the error in the definition of the enumeration.

CCN6283 ″%1$s″ is not a viable candidate.

Where: ″%1$s″ is the potential resolution.

Explanation: Informational message indicating that this was not a viable candidate for overload resolution.

User Response: See the primary message.

CCN6284 Predefined ″%1$s″ is not a viable candidate.

Where: ″%1$s″ is the potential resolution.

Explanation: Informational message indicating that this was not a viable candidate for overload resolution.

User Response: See the primary message.

CCN6285 The specialization matches ″%1$s″.

Where: ″%1$s″ is the matched specialization.

Explanation: Informational message indicating what a specialization matches.

User Response: See the primary message.

CCN6286 The specialization does not match ″%1$s″.

Where: ″%1$s″ is what the specialization cannot match.

Explanation: Informational message indicating what a specialization cannot match.

User Response: See the primary message.

CCN6287 ″%1$s″ has internal linkage but is undefined.

Where: ″%1$s″ is the undefined member variable or static function.

Explanation: A static member variable or static function must be defined.

User Response: Define the member variable or static function.

CCN6288 The explicit template instantiation ″%1$s″ matches more than one template.

Where: ″%1$s″ is the explicit instantiation.

Explanation: The explicit instantiation of this function matches multiple function templates. Probably because of allowable non-type template argument conversions.

User Response: Remove the explicit instantiation, remove one of the primary templates, or add namespaces to separate the templates.

CCN6289 The implicit object parameter of type ″%2$s″ cannot be initialized with an implied argument of type ″%1$s″.

Where: ″%2$s″ is the implicit object parameter type. ″%1$s″ is the implied argument type.

Explanation: A function is being called implicitly and the parameters do not match the expected parameters.

User Response: Provide an explicit conversion function.

CCN6290 An rvalue cannot be converted to a reference to a non-const type.

Explanation: Informational message indicating that the target of the conversion must be const.

User Response: See the primary message.

CCN6291 To initialize the reference with an rvalue, ″%1$s″ must have a copy constructor with a parameter of type ″%2$s″.

Where: ″%1$s″ is the type of the object. ″%2$s″ is the type of the parameter.

Explanation: Informational message indicating that a copy constructor must be supplied.

User Response: See the primary message.

CCN6292 Static declarations are not considered for a function call if the function is not qualified.

Explanation: Informational message describing why a static function cannot be considered.

User Response: See the primary message.

CCN6293 The explicit instantiation matches ″%1$s″.

Where: ″%1$s″ is the matched explicit instantiation.

Explanation: Informational message about matching of explicit instantiations.

User Response: See the primary message.

CCN6294 The explicit instantiation does not match ″%1$s″.

Where: ″%1$s″ is the explicit instantiation that is not matched.

Explanation: Informational message about matching of explicit instantiations.

User Response: See the primary message.

CCN6295 The explicit template instantiation ″%1$s″ does not match any template.

Where: ″%1$s″ is the explicit template instantiation.

Explanation: There is no primary template matching this explicit template instantiation.

User Response: Remove the explicit template instantiation or declare the primary template..

CCN6296 The const object ″%1$s″ requires ″%2$s″ to have a user-declared default constructor.

Where: ″%1$s″ is the const object. ″%2$s″ is the class.

Explanation: This class has a const object so the class must have a user-declared default constructor.

User Response: Provide a user default-constructor.

CCN6297 The const object ″%1$s″ needs an initializer or requires ″%2$s″ to have a user-declared default constructor.

Where: ″%1$s″ is the const object. ″%2$s″ is the class.

Explanation: This class has a const object so the class must have a user-declared default constructor.

User Response: Provide a user default-constructor.

CCN6298 ″%1$s″ needs to be declared in the containing scope to be found by name lookup.

Where: ″%1$s″ is the class.

Explanation: Informational message about declaring friend classes in the containing scope for the class to be found by name lookup.

User Response: Declare the class in the enclosing scope.

CCN6299 ″%1$s″ is undefined. Every variable of type ″%2$s″ will assume ″%3$s″ has no virtual bases and does not use multiple inheritance.

Where: ″%1$s″ is the undefined class. ″%2$s″ is the pointer type. ″%3$s″ is the class.

Explanation: The pointer refers to an incomplete class so it will be assumed that the class has no virtual bases nor multiple inheritance.

User Response: Define the class.

CCN6300 ″%1$s″ includes the file ″%2$s″.

Where: ″%1$s″ and ″%2$s″ are the two files in the include chain.

Explanation: This is a submessage. This message is used to specify that a certain file includes the file ″%2$s″.

User Response: See the primary message.

CCN6301 The previous error occurs during the processing of file ″%1$s″.

Where: ″%1$s″ is the file.

Explanation: This is a submessage.

User Response: See the primary message.

CCN6302 The conflicting declaration was encountered during the processing of the file ″%1$s″.

Where: ″%1$s″ is the file name.

Explanation: This message describes the include hierarchy that caused the preceding error.

User Response: Remove the conflicting declaration.

CCN6303 ″%1$s″ is not visible.

Where: ″%1$s″ is the declaration.

Explanation: This message indicates that the declaration is not visible at the current location.

User Response: Move the declaration to a position prior to the current location.

CCN6304 ″%1$s″ is not visible from ″%2$s″.

Where: ″%1$s″ is the declaration. ″%2$s″ is the location.

Explanation: This message indicates that the declaration is not visible at the current location.

User Response: Move the declaration to a position prior to the current location.

CCN6305 ″%1$s″ is not complete when included by ″%2$s″.

Where: ″%1$s″ is the class. ″%2$s″ is the header file.

Explanation: The class or struct is incomplete when included from a particular header file location.

User Response: Instantiate the direct nullifier of the virtual function table operator.

CCN6400 The incorrect #pragma is ignored.

Explanation: The pragma is incorrect and is ignored.

User Response: Correct the pragma.

CCN6401 An unknown ″#pragma %1$s″ is specified.

Where: The name of the unknown pragma.

Explanation: The specified pragma is not recognised.

User Response: Change the name of the pragma to one that is applicable to the compiler.

CCN6402 The options for ″#pragma %1$s″ are incorrectly specified: expected %2$s and found %3$s. The pragma is ignored.

Where: The name of the pragma and the expected and found options.

Explanation: The options for the pragma are not correctly specified and the pragma is ignored.

User Response: Change the options to the pragma as indicated.

CCN6403 The function ″%2$s″ specified in ″#pragma %1$s″ cannot be found. The pragma is ignored.

Where: The names of the pragma and the undeclared function, respectively.

Explanation: The pragma is ignored because it refers to a function that is not declared.

User Response: Change the pragma to refer to a declared function or declare the function.

CCN6404 The parameter ″%1$s″ specified for ″#pragma %2$s″ is not valid. The pragma is ignored.

Where: The invalid parameter and the pragma, respectively.

Explanation: The pragma is ignored because the parameter specified is not valid.

User Response: Change the pragma parameter.

CCN6405 Syntax error in ″#pragma %1$s″: expected %2$s and found %3$s. The pragma is ignored.

Where: The name of the pragma, the expected text and the incorrect input, respectively.

Explanation: The pragma is ignored because there is a syntax error in the pragma directive.

User Response: Correct the syntax of the pragma specification.

CCN6406 ″#pragma %1$s″ is already specified. The pragma is ignored.

Where: The name of the pragma that is ignored.

Explanation: The pragma is ignored because it has already been specified.

User Response: Remove the pragma specification.

CCN6407 The function ″%2$s″ specified in ″#pragma %1$s″ does not have an implementation. The pragma is ignored.

Where: The name of the ignored pragma and the name of the function that must be defined.

Explanation: The pragma is ignored because it requires that the specified function be defined but it is only declared.

User Response: Define the function.

CCN6408 ″#pragma %1$s″ has no effect. The pragma is ignored.

Where: The name of the ignored pragma.

Explanation: Informational message that the pragma is ignored because it has no effect. It may be that the pragma specifies options that are already in effect.

User Response: See the primary message.

CCN6409 ″#pragma %1$s″ is not supported on the target platform. The pragma is ignored.

Where: The name of the ignored pragma.

Explanation: Informational message that the pragma is ignored because it is not valid on the target platform.

User Response: See the primary message.

CCN6410 The function ″%2$s″ specified in ″#pragma %1$s″ is an overloaded function. The pragma is ignored.

Where: The name of the ignored pragma and the name of the overloaded function, respectively.

Explanation: The pragma is ignored because the function specified is overloaded so it is not clear which function is being specified.

User Response: Remove the pragma or ensure that the function is not overloaded.

CCN6411 ″#pragma %1$s″ must be specified in global scope. The pragma is ignored.

Where: The name of the ignore pragma.

Explanation: The pragma is ignored because it has been specified in an invalid scope such as a function body or class member list.

User Response: Move the pragma to global scope.

CCN6412 The declaration ″%2$s″ specified in ″#pragma %1$s″ cannot be found. The pragma is ignored.

Where: The name of the ignored pragma and the name of the variable or type indicated in the pragma.

Explanation: The pragma is ignored because it names a variable or type that has not been declared.

User Response: Change the pragma to refer to a declared variable or type or declare the indicated variable or type.

CCN6413 The conflicting pragma is specified on line %1$s of ″%2$s″.

Where: The coordinates of the conflicting pragma.

Explanation: Informational message about the coordinates of the conflicting pragma.

User Response: See the primary message.

CCN6414 The function ″%2$s″ specified in ″#pragma %1$s″ is a member function. The pragma is ignored.

Where: ″%1$s″ is the pragma name. ″%2$s″ is the function name.

Explanation: Member functions are not allowed for the pragma specified.

User Response: Specify a non-member function in the pragma or remove the pragma.

CCN6415 The declaration ″%2$s″ specified in ″#pragma %1$s″ is a member variable. The pragma is ignored.

Where: ″%1$s″ is the pragma name. ″%2$s″ is the declaration.

Explanation: Member variables are not allowed for the pragma specified.

User Response: Specify a non-member variable in the pragma or remove the pragma.

CCN6416 The declaration ″%2$s″ specified in ″#pragma %1$s″ is a structure tag. The pragma is ignored.

Where: ″%1$s″ is the pragma name. ″%2$s″ is the declaration.

Explanation: Structure tags are not allowed for the pragma specified.

User Response: Fix the declaration in the pragma or remove the pragma.

CCN6417 The declaration ″%2$s″ specified in ″#pragma %1$s″ must have ″%3$s″ linkage. The pragma is ignored.

Where: ″%1$s″ is the pragma name. ″%2$s″ is the declaration specified in the pragma. ″%3$s″ is the required linkage for the pragma.

Explanation: The pragma is only valid for declarations with specific linkage.

User Response: Specify a declaration with the correct linkage or remove the pragma.

CCN6418 The function ″%1$s″ specified in #pragma ″%2$s″ is not compatible with the function ″%3$s″, which is also specified in the #pragma. The pragma will be ignored.

Where: %1$s, and %3$s are functions, %2$s is the pragma name.

Explanation: The two functions specified in the pragma are incompatibile.

User Response: Change the functions or remove the pragma.

CCN6420 The packing boundary for ″#pragma pack″ must be 1, 2, 4, 8, or 16. The pragma is ignored.

Explanation: A ″#pragma pack″ has been specified with an invalid boundary.

User Response: Change the pack boundary for the ″#pragma pack″ to one of the accepted boundaries or remove the pragma.

CCN6421 The ″#pragma pack″ stack is empty. The current alignment may change.

Explanation: The current alignment may change because the stack for the #pragma pack is empty.

User Response: Remove the pragma or ensure that the pragma stack is not empty by making sure that there are an appropriate number of push pragmas.

CCN6422 The identifier does not exist within the ″#pragma pack″ stack. The current alignment may change.

Explanation: The current alignment may change because the identifier does not exist on the #pragma pack stack.

User Response: Change the name of the identifier specified in the pragma.

CCN6423 The declaration in ″#pragma map″ has already been mapped to ″%1$s″. The pragma is ignored.

Where: ″%1$s″ is the previous mapping of the declaration.

Explanation: The pragma is ignored because the declaration has already been mapped.

User Response: Remove the pragma or change the declaration.

CCN6424 Priority values in successive ″#pragma priority″ statements must increase.

Explanation: The priority specified is lower than a priority specified in a previous pragma.

User Response: Increase the priority specified in the pragma.

CCN6425 The value given for the ″#pragma priority″ is in the range reserved for the system.

Explanation: The priority specified in the pragma is in the range reserved for the system. This may cause unexpected behaviour because the declaration may have a higher priority than system variables.

User Response: Lower the specified priority.

CCN6426 The function ″%1$s″ in ″#pragma alloc_text″ is already specified. The pragma is ignored.

Where: The name of the function specified in the pragma.

Explanation: The pragma is ignored because the function has already been specified in a previous #pragma alloc_text.

User Response: Remove the pragma.

CCN6427 The specified object model ″%1$s″ is not known. The pragma is ignored.

Where: The unrecognised object model.

Explanation: The pragma is ignored because the object model is not recognised.

User Response: Change the specified object model to one that is known.

CCN6428 The ″#pragma object_model″ stack is empty. The pragma is ignored.

Explanation: The pragma is ignored because the object model stack is empty.

User Response: Remove the pragma or ensure that the stack is not empty.

CCN6429 The identifier ″%1$s″ in ″#pragma import″ is already specified on line %2$s of ″%3$s″. The pragma is ignored.

Where: The name of the repeated identifier and the coordinates of the previous pragma.

Explanation: The pragma is ignored because the identifier has already been specified in a previous #pragma import.

User Response: Remove the pragma.

CCN6430 The identifier ″%1$s″ in ″#pragma export″ is already specified. The pragma is ignored.

Where: The name of the repeated identifier and the coordinates of the previous pragma.

Explanation: The pragma is ignored because the identifier has already been specified in a previous #pragma export.

User Response: Remove the pragma.

CCN6431 The ″#pragma enum″ stack is empty. The pragma is ignored.

Explanation: The pragma is ignored because the pragma enum stack is empty.

User Response: Remove the pragma or ensure that the pragma stack is not empty.

CCN6432 The function ″%1$s″ in ″#pragma alloc_text″ is already specified with ″#pragma code_seg″.

Where: The name of the function indicated in the pragma.

Explanation: The pragma is in conflict with a previous #pragma code_seg.

User Response: Remove the current or the previous pragma.

CCN6433 The function ″%1$s″ in ″#pragma weak″ is already specified. The pragma is ignored.

Where: The name of the function specified in the pragma.

Explanation: The pragma is ignored because it has already been specified in a #pragma weak.

User Response: Remove the pragma.

CCN6434 The message id ″%1$s″ in ″#pragma report″ is not a valid. The pragma is ignored.

Where: The message id that must be changed.

Explanation: The pragma is ignored because the message id is not valid.

User Response: Change the message id.

CCN6435 The function ″%1$s″ in ″#pragma mc_func″ is already specified. The pragma is ignored.

Where: The name of the function specified in the pragma.

Explanation: The pragma is ignored because the function has already been specified in a #pragma mc_func.

User Response: Remove the pragma.

CCN6436 The function ″%1$s″ in ″#pragma reg_killed_by″ is already specified. The pragma is ignored.

Where: The name of the function specified in the pragma.

Explanation: The pragma is ignored because the function has already been specified in a #pragma reg_killed_by.

User Response: Remove the pragma.

CCN6437 ″#pragma reg_killed_by″ must be used with a corresponding ″#pragma mc_func″.

Explanation: The function specified in the pragma must have been previous specified in a #pragma mc_func.

User Response: Provide the #pragma mc_func before the #pragma reg_killed_by.

CCN6438 The file ″%1$s″ should be specified in an ″#include″ directive or as a source file in the configuration file.

Where: The name of the file that should be included.

Explanation: Informational message indicating that the file should be an included file or it should be specified in the configuration file.

User Response: Ensure that the file is specified in an include directive.

CCN6439 Two or more expressions must be specified in ″#pragma disjoint″. The pragma is ignored.

Explanation: The pragma is ignored because it must have two or more expressions specified.

User Response: Ensure that at least two expressions are specified in the pragma.

CCN6440 The expressions ″%1$s″ and ″%2$s″ specified in ″#pragma disjoint″ have incompatible types. The pragma is ignored.

Where: The two incompatible expressions, one of which must be changed.

Explanation: The pragma is ignored because the types specified in the two expressions are incompatible.

User Response: Change one of the expressions to have a compatible type with the other.

CCN6441 The expression ″%1$s″ specified in ″#pragma disjoint″ is not a valid type. The pragma is ignored.

Where: The expression specifying the invalid type.

Explanation: The pragma is ignored because the type specified in the expression is not correct.

User Response: Change the expression to specify a valid type.

CCN6442 The ″#pragma align″ stack is empty. The pragma is ignored.

Explanation: The pragma is ignored because the #pragma align stack is empty.

User Response: Remove the pragma or ensure that the #pragma align stack is not empty.

CCN6443 ″#pragma %1$s″ overrides the original option value.

Where: The name of the pragma that is overriding the option value.

Explanation: Informational message indicating that the pragma is overriding the option value.

User Response: See the primary message.

CCN6444 The ″#pragma namemangling″ stack is empty. The pragma is ignored.

Explanation: The pragma is ignored because the #pragma namemangling stack is empty.

User Response: Remove the pragma or ensure that the #pragma namemangling stack is not empty.

CCN6445 The size specified for ″#pragma pointer_size″ must be 32 or 64. The pragma is ignored.

Explanation: The pragma is ignored because the size specified was not 32 or 64.

User Response: Change the size specified to be 32 or 64.

CCN6446 The ″#pragma pointer_size″ stack is empty.

Explanation: The pragma is ignored because the #pragma pointer_size stack is empty.

User Response: Remove the pragma or ensure that the #pragma pointer_size stack is not empty.

CCN6447 The argument ″%2$s″ specified in ″#pragma %1$s″ is not a defined class.

Where: ″%1$s″ is the name of the pragma. ″%2$s″ is the name of the class that must be defined.

Explanation: The pragma is ignored because the argument does not specify a defined class.

User Response: Change the argument or ensure that the class is defined.

CCN6448 ″A #pragma IsHome″ is defined for ″%1$s″, but there is no matching ″#pragma HasHome″. The pragma is ignored.

Where: The argument that must have a corresponding #pragma HasHome.

Explanation: The pragma is ignored because there must be a previously specified #pragma HasHome for the argument.

User Response: Remove the pragma or ensure that there is a previous corresponding #pragma HasHome.

CCN6449 More than one ″#pragma IsHome″ for ″%1$s″ in different targets.

Where: The argument that has multiple #pragma IsHome directives.

Explanation: There are more than one #pragma IsHome specified for the arguments in different targets.

User Response: Remove the extra #pragma IsHome directives.

CCN6450 ″%1$s″ is not a valid .ini file directive.

Where: The invalid directive that must be changed.

Explanation: The directive is not valid for an .ini file.

User Response: Change the directive.

CCN6451 The first directive in an .ini file must be a category line.

Explanation: The format of the .ini file is incorrect. A category line was not found first.

User Response: Change the first directive to be a category line.

CCN6452 The category [%1$s] already appears in the .ini file ″%2$s″.

Where: ″%1$s″ is the category that is repeated. ″%2$s″ is the name of the .ini file.

Explanation: The category has already been specified in a .ini file.

User Response: Remove the specification.

CCN6453 The property ″%1$s″ is already listed in category [%2$s] in the .ini file ″%3$s″.

Where: ″%1$s″ is the property that is repeated. ″%2$s″ is the category in which it is repeated. ″%3$s″ is the name of the .ini file.

Explanation: There is a problem with the syntax of the .ini file. A property is duplicated in two categories.

User Response: Remove the duplicate property from the .ini file.

CCN6454 The declaration ″%1$s″ specified in #pragma ″%2$s″ for communications area was not resolved or is invalid.

Where: ″%1$s″ is the declaration in error. ″%2$s″ is the name of the pragma.

Explanation: This is an OS/400 (iSeries) message.

User Response: Fix the declaration or remove the pragma.

CCN6455 Member ″%1$s″ is not declared as specified in #pragma ″%2$s″. The pragma is ignored.

Where: ″%1$s″ is the class member. ″%2$s″ is the name of the pragma.

Explanation: The declaration of the member in the pragma does not match the declaration for that member in the member's class.

User Response: Fix the declaration in the pragma or remove the pragma.

CCN6456 Only dot member access is allowed in #pragma ″%1$s″. The pragma is ignored.

Where: ″%1$s″ is the name of the pragma.

Explanation: Pragma ″%1$s″ is only allowed to use class member access with the dot operator.

User Response: Change the pragma to use dot member access or remove the pragma.

CCN6457 Member ″%1$s″ is at offset ″%2$s″, not at offset ″%3$s″ as specified in #pragma assert_field_offset.

Where: ″%1$s″ is the member name. ″%2$s″ is the actual offset. ″%3$s″ is the offset specified in the pragma.

Explanation: The assertion in the #pragma assert_field_offset has been violated. The member is not at the specified offset.

User Response: Fix the offset or remove the pragma.

CCN6458 The name ″%1$s″ specified in #pragma argopt is not a function, function pointer, function typedef, or function pointer typedef. The pragma will be ignored.

Where: ″%1$s″ is the name specified in the pragma.

Explanation: The #pragma argopt only applies to functions. This is an OS/400 (iSeries) message.

User Response: Specify a valid name or remove the pragma.

CCN6459 The function ″%1$s″ specified in #pragma argopt has a variable length argument list. The pragma will be ignored.

Where: ″%1$s″ is the function name.

Explanation: The #pragma argopt cannot be used with a function that uses an ellipsis in its parameter list. This is an OS/400 (iSeries) message.

User Response: Specify a function without a variable length argument list or remove the pragma.

CCN6460 ″%1$s″ has not been declared before the pragma pointer directive.

Where: ″%1$s″ is the type.

Explanation: ″%1$s″ must be declared before the pragma.

User Response: Add a declaration for ″%1$s″ before the pragma or remove the pragma.

CCN6461 ″%1$s″ is not a void pointer.

Where: ″%1$s″ is the argument to the pragma.

Explanation: The pragma has an argument.

User Response: Fix the argument to the pragma or remove the pragma.

CCN6462 ″%1$s″ is not a valid ILE pointer type.

Where: ″%1$s″ is the argument to the pragma.

Explanation: This is an OS/400 (iSeries) message.

User Response: ″Fix the argument to the pragma or remove the pragma.

CCN6463 ″%1$s″ has been used in a declaration, the pragma is ignored.

Where: ″%1$s″ is the argument to the pragma.

Explanation: The name has already been used previously and cannot be used again by the pragma.

User Response: Fix the argument to the pragma or remove the pragma.

CCN6464 ″%1$s″ is not a typedef name.

Where: ″%1$s″ is the argument to the pragma.

Explanation: The pragma requires a typedef name as an argument and the one provided is not one.

User Response: Fix the argument to the pragma or remove the pragma.

CCN6465 Instruction sequence for ″#pragma mc_func″ contains the character ″%1$s″ that is not a hexadecimal digit.

Where: ″%1$s″ is the invalid character specified in the pragma.

Explanation: The pragma requires a hexadecimal argument and one has not been provided.

User Response: Fix the instruction sequence for the pragma or remove the pragma.

CCN6466 Instruction sequence for ″#pragma mc_func″ contains odd number of hexadecimal digits.

Explanation: The pragma requires an argument which is an instruction sequence consisting of an even number of hexadecimal digits.

User Response: Fix the instruction sequence for the pragma or remove the pragma.

CCN6467 The include directive for the primary source file ″%1$s″ is ignored.

Where: ″%1$s″ is the source file name.

Explanation: It was not possible for the compiler to process the file as a primary source file.

User Response: Remove the include directive from the configuration file.

CCN6468 The function ″%1$s″ specified in ″#pragma %2$s″ has ″%3$s″ linkage. The pragma is ignored.

Where: ″%1$s″ is the function name. ″%2$s″ is the name of the pragma.

Explanation: This is an OS/400 (iSeries) message.

User Response: Specify a function with the correct linkage or remove the pragma.

CCN6469 The function ″%2$s″ specified in ″#pragma %1$s″ cannot be found.

Where: ″%1$s″ is the name of the pragma. ″%2$s″ is the function name.

Explanation: Name lookup failed for the function specified in the pragma.

User Response: Fully qualify the function, specify a different function, or remove the pragma.

CCN6470 The source file ″%1$s″ is being included by the source file ″%2$s″, which has different options in effect.

Where: ″%1$s’ is the included source file. ″%2$s″ is the source file including ″%1$s″

Explanation: The source file ″%1$s″ has been specified as a primary source file in the configuration file and it’s options do not match the options specified by another primary source file that includes ″%1$s″.

User Response: Change the options to be consistent or change ″1s″ to not be a primary source file.

CCN6471 The function or label ″%1$s″ specified in ″#pragma exception_handler″ was not resolved or is invalid.

Where: ″%1$s″ is the function or label.

Explanation: This is an OS/400 (iSeries) message.

User Response: Fix the argument to the pragma or remove the pragma.

CCN6472 The expression ″%1$s″ specified in ″#pragma exception_handler″ for parameter ″%2$s″ is not a valid type.

Where: ″%1$s″ is the expression. ″%2$s″ is the parameter to the pragma.

Explanation: This is an OS/400 (iSeries) message.

User Response: Fix the expression or remove the pragma.

CCN6473 The expression ″%1$s″ specified in ″#pragma exception_handler″ for parameter ″%2$s″ is a non-const or non-integral expression.

Where: ″%1$s″ is the expression. ″%2$s″ is the parameter to the pragma.

Explanation: This is an OS/400 (iSeries) message.

User Response: Fix the expression or remove the pragma.

CCN6474 The value for control action parameter expression ″%1$s″ is only valid for a function handler (not a label as was given or interpreted).

Where: ″%1$s″ is the expression specified in the pragma.

Explanation: This is an OS/400 (iSeries) message.

User Response: Fix the expression or remove the pragma.

CCN6475 The value for control action parameter expression ″%1$s″ is not valid.

Where: ″%1$s″ is the expression specified in the pragma.

Explanation: This is an OS/400 (iSeries) message.

User Response: Fix the expression or remove the pragma.

CCN6476 Invalid message identifier ″%1$s″ in message ID list parameter on ″#pragma exception_handler″.

Where: ″%1$s″ is the invalid message identifier specified in the pragma.

Explanation: This is an OS/400 (iSeries) message.

User Response: Fix the message identifier or remove the pragma.

CCN6477 Invalid message identifier list ″%1$s″ on ″#pragma exception_handler″.

Where: ″%1$s″ is the invalid message identifier list specified in the pragma.

Explanation: This is an OS/400 (iSeries) message.

User Response: Fix the message identifier list or remove the pragma.

CCN6478 The function ″%1$s″ specified in ″%2$s″ was not resolved to a correctly defined and prototyped function.

Where: ″%1$s″ is the function name. ″%2$s″ is the name of the pragma.

Explanation: Name lookup for ″%1$s″ failed. This is an OS/400 (iSeries) message.

User Response: Provide a declaration for ″%1$s″ or remove the pragma.

CCN6479 Disable handler has no matching cancel/exception handler, or cancel/exception handler is out of scope.

Explanation: This is an OS/400 (iSeries) message.

User Response: Add a matching cancel/exception handler or remove the pragma.

CCN6480 Function ″%1$s″ has not been declared before the pragma descriptor directive.

Where: ″%1$s″ is the function name argument to the pragma.

Explanation: This is an OS/400 (iSeries) message.

User Response: Declare ″%1$s″ before the pragma descriptor directive or remove the pragma.

CCN6481 Operational descriptors for type ″%1$s″ not supported.

Explanation: This is an OS/400 (iSeries) message.

CCN6482 Function cannot have C++ or OS linkage.

Explanation: This is an OS/400 (iSeries) message.

CCN6483 ″void″ expected, but found ″%1$s″; operational descriptor for return type not currently supported.

Where: ″%1$s″ is the unexpected text.

Explanation: This is an OS/400 (iSeries) message.

CCN6484 More parameters than the function prototype.

Explanation: This is an OS/400 (iSeries) message.

CCN6485 Invalid operational descriptor specifier ″%1$s″.

Where: %1$s″ is the operational descriptor specifier.

Explanation: This is an OS/400 (iSeries) message.

CCN6486 Descriptor specifier ″%1$s″ invalid for type ″%2$s″.

Explanation: This is an OS/400 (iSeries) message.

CCN6492 No argument is specified for ″#pragma define″. The pragma is ignored.

Explanation: The pragma requires an argument and one was not specified.

User Response: Specify an argument or remove the pragma.

CCN6493 Duplicate argument ″%1$s″ in ″#pragma disjoint″. The pragma is ignored.

Where: ″%1$s″ is the duplicate argument specified in the pragma.

Explanation: The argument indicated was duplicated in the argument list specified for the pragma.

User Response: Remove the duplicate argument or remove the pragma.

CCN6494 The suboption ″%1$s″ for ″#pragma %2$s″ is not supported on the target platform. The pragma is ignored.

Where: ″%1$s″ is the name of the suboption that is unsupported. ″%2$s″ is the name of the pragma.

Explanation: The suboption for the pragma indicated is not supported on this operating system.

User Response: Specify a different suboption or remove the pragma.

CCN6495 Unexpected text ″%2$s″ found in ″#pragma %1$s″. The pragma is ignored.

Where: ″%1$s″ is the name of the pragma. ″%2$s″ is the text causing the syntax error.

Explanation: A syntax error has been found while processing the pragma, causing it to be ignored.

User Response: Fix the syntax of the pragma or remove the pragma.

CCN6496 Unexpected text ″%2$s″ found in ″#pragma %1$s″. The rest of the pragma directive is ignored.

Where: ″%1$s″ is the name of the pragma. ″%2$s″ is the text causing the syntax error.

Explanation: A syntax error has been found while processing part of the pragma, causing part of it to be ignored.

User Response: Fix the syntax of the pragma or remove the pragma.

CCN6497 An implicit ″}″ does not find a matching implicit ’extern ″C″ {’. An extra ″}″ may be present.

Explanation: An unmatched ″}″ was detected while processing a linkage specification.

User Response: Remove the extra ″}″ if one exists.

CCN6498 The function ″%2$s″ specified in ″#pragma %1$s″ has already been defined. The pragma is ignored.

Where: ″%1$s″ is the name of the pragma. ″%2$s″ is the name of the function.

Explanation: The pragma specified must be placed before the definition of the function to which it refers.

User Response: Move the pragma to before the definition of the function or remove the pragma.

CCN6499 The function ″%2$s″ specified in ″#pragma %1$s″ is virtual. The pragma is ignored.

Where: ″%1$s″ is the name of the pragma. ″%2$s″ is the name of the function.

Explanation: The pragma specified requires an argument that is not a virtual function.

User Response: Change the pragma to specify a non-virtual function or remove the pragma.

CCN6600 ″main″ should have a return type of type ″int″.

Explanation: A return type other than ″int″ has been specified for ″main″.

User Response: Change the return type of ″int″ to be ″int″.

CCN6601 A local class cannot have member templates.

Explanation: Member templates can only be defined in namespace scope classes.

User Response: Remove the template from the local class, or move the class to non-local scope.

CCN6602 The partial specialization ″%1$s″ cannot have template parameters that have default values.

Where: ″%1$s″ is the partial specialization.

Explanation: Default template arguments are not allowed on partial specializations.

User Response: Remove the default template arguments.

CCN6603 Default template parameter arguments cannot be followed by uninitialized template parameters.

Explanation: Just like function parameters, all template parameters following a template parameter with a default argument must also have default arguments.

User Response: Add the missing default arguments or remove the existing one.

CCN6604 The template parameter ″%1$s″ cannot be used in a partially specialized non-type argument expression.

Where: ″%1$s″ is the template parameter.

Explanation: The use of a template parameter in an expression for a non-type template argument in partial specialization is not allowed.

User Response: Correct the non-type template argument expression.

CCN6605 The argument list for the partial specialization ″%1$s″ is equivalent to the implicit argument list of the primary template.

Where: ″%1$s″ is the partial specialization.

Explanation: A partial specialization must specialize something in the argument list.

User Response: Change the argument list of the partial specialization.

CCN6606 A non-type template parameter ″%1$s″ must have integral, enumeration, pointer, reference, or pointer-to-member type.

Where: ″%1$s″ is the non-type template parameter.

Explanation: No other types are allowed.

User Response: Correct the non-type template parameter type.

CCN6607 All array dimensions for ″%1$s″ should be specified and should be greater than zero.

Where: ″%1$s″ is the array.

Explanation: An array dimension is missing or is negative.

User Response: Ensure that all dimensions are specified as non-negative numbers.

CCN6608 An anonymous %1$s should only define non-static data members.

Where: %1$s is the keyword union, struct, or class.

Explanation: Static members are not allowed in anonymous aggregates.

User Response: Remove the static member declaration.

CCN6609 A using declaration cannot be used to declare ″%1$s″.

Where: ″%1$s″ is the declarator.

Explanation: The using declaration cannot be used here.

User Response: Remove the using declaration.

CCN6610 ″%1$s″ must not be declared as import and defined.

Where: ″%1$s″ is the function.

Explanation: The ″_Import″ specifier cannot be specified on a definition.

User Response: Remove the ″_Import″ specifier.

CCN6611 The current option settings do not allow the use of ″long long″.

Explanation: The declaration type is ″long long″ but this type is disallowed due to option settings.

User Response: Change the type of the declaration or the option settings to allow ″long long″.

CCN6612 The static variable ″%1$s″ is not visible where ″%2$s″ is used in a #include directive.

Where: ″%1$s″ is the static variable. ″%2$s″ is the header file.

Explanation: A static variable is being referenced in an include file and is not visible.

User Response: Remove the static specifier from the declaration.

CCN6613 The static function ″%1$s″ is not visible where ″%2$s″ is used in a #include directive.

Where: ″%1$s″ is the static function. ″%2$s″ is the header file.

Explanation: A static function is being referenced in an include file and is not visible.

User Response: Remove the static specifier from the declaration.

CCN6614 ″%1$s″ must be the last data member in its class because ″%2$s″ contains a zero-dimension array.

Where: ″%1$s″ is the member. ″%2$s″ is the union, struct, or class.

Explanation: Only the last non-static data member can have a zero dimension.

User Response: Move the declaration to be the last in the class.

CCN6615 Only the first array bound can be omitted.

Explanation: For a multi-dimensional array, the compiler can determine the size of the first bound based on the number of initializers. It is unable to compute any other omitted array bounds.

User Response: Specify all array bounds or leave only the first bound unspecified.

CCN6616 A pointer-to-member should not be converted from the virtual base ″%1$s″ to the derived class ″%2$s″.

Where: ″%1$s″ is the virtual base. ″%2$s″ is the derived class.

Explanation: The conversion is from a virtual base class to a derived class.

User Response: See the primary message.

CCN6617 The incomplete type ″%1$s″ is not allowed in an exception-specification .

Where: ″%1$s″ is the incomplete type.

Explanation: Only complete types are allowed in an exception-specification.

User Response: Correct the exception specification type list.

CCN6618 ″%1$s″ is not allowed in an exception-specification because ″%2$s″ is incomplete.

Where: ″%1$s″ is the pointer type. ″%2$s″ is the incomplete type.

Explanation: Only pointers to complete types are allowed in pointer exception-specification types.

User Response: Correct the exception-specification type list.

CCN6619 The type ″%1$s″ is not valid in this context.

Where: ″%1$s″ is the type.

Explanation: The type ″void″ is not valid for this declaration.

User Response: Change the type.

CCN6620 ″%1$s″ must be declared to have ″stdcall″ linkage.

Where: ″%1$s″ is the function.

Explanation: The ″stdcall″ specifier must be specified.

User Response: Add the ″stdcall″ specifier.

CCN6621 The explicit specialization ″%1$s″ must be declared in the namespace containing the template.

Where: ″%1$s″ is the explicit specialization.

Explanation: The primary template and an explicit specialization declaration must be in the same scope.

User Response: Move the explicit specialization declaration to the correct scope.

CCN6622 The explicit specialization ″%1$s″ must be defined in a namespace that encloses the declaration of the explicit specialization.

Where: ″%1$s″ is the explicit specialization.

Explanation: An explicit specialization must be defined at namespace scope, in the same or an enclosing namespace as the declaration.

User Response: Move the explicit specialization definition to the correct scope.

CCN6623 The explicit specialization ″%1$s″ cannot have default function arguments.

Where: ″%1$s″ is the explicit specialization.

Explanation: Default function arguments are not allowed on an explicit specialization.

User Response: Remove the default function arguments.

CCN6624 The partial specialization ″%1$s″ must be declared in the same scope as the primary template or in a namespace scope that encloses the primary template.

Where: ″%1$s″ is the partial specialization.

Explanation: A partial specialization declaration must be in the same scope or in an enclosing namespace scope of the primary template.

User Response: Move the partial specialization declaration to the correct scope.

CCN6625 The explicit specialization ″%1$s″ must not be declared in the scope of a template.

Where: ″%1$s″ is the explicit specialization.

Explanation: An explicit specialization must be declared in the namespace containing the primary template.

User Response: Remove the explicit specialization.

CCN6626 At least one template argument in a partial specialization must depend on a template parameter.

Explanation: A partial specialization cannot be fully specialized.

User Response: Change the declaration to an explicit specialization or change the template arguments to be partially specialized.

CCN6627 The bit-field ″%1$s″ cannot be greater than 32 bits.

Where: ″%1$s″ is the bit-field.

Explanation: The size of the bit-field is too large.

User Response: Use a smaller size for the bit-field.

CCN6628 Every template parameter for a constructor template must be used in the parameter list of the constructor.

Explanation: There is no way to specify an explicit template argument list for a constructor template.

User Response: Change the template parameter list of the constructor template.

CCN6629 Every template parameter for a conversion function template must be used in the return type.

Explanation: There is no way to specify an explicit template argument list for a conversion function template.

User Response: Change the template parameter list of the conversion function template

CCN6630 Every template parameter for a partial specialization must be used in the template argument list.

Explanation: The extra template parameters are not used so they are not allowed.

User Response: Change the parameter list of the partial specialization.

CCN6631 This message is not used.

CCN6632 The length of the identifier exceeds the maximum limit of ″%1$s″ for a name with ″%2$s″ linkage.

Where: ″%1$s″ is the maximum permitted identifier length. ″%2$s″ is the linkage specifier.

Explanation: The identifier name is too large.

User Response: Replace the identifier with a smaller identifier.

CCN6633 The name ″%1$s″ is not a recognized built-in declaration.

Where: ″%1$s″ is the function name.

Explanation: The function specified is not a built-in function.

User Response: Change the declaration so that it does not specify that the function is built in.

CCN6634 An array element must not have type ″%1$s″.

Where: ″%1$s″ is the type.

Explanation: The type of the array is invalid.

User Response: Change the type of the array.

CCN6635 There cannot be a reference to a reference.

Explanation: A reference to a reference is invalid.

User Response: Remove the extra reference.

CCN6636 There cannot be a pointer to a reference.

Explanation: A pointer to a reference is invalid.

User Response: Change the declaration.

CCN6637 There cannot be a pointer-to-member with reference type.

Explanation: A pointer to a member reference is invalid.

User Response: Change the declaration.

CCN6638 There cannot be an array of references.

Explanation: The element type of an array cannot be a reference type, void type, function type, or an abstract class type.

User Response: Change the element type of the array to a valid type.

CCN6639 When 64-bit mode is implemented, the behavior of long type bit-fields may change. Long type bit-fields currently default to int.

Explanation: The bit-field will default to int, but this behavior may change in future 64-bit modes.

User Response: None.

CCN6640 Cannot take the address of the machine-coded function ″%1$s″.

Where: ″%1$s″ is the function.

Explanation: It is invalid to take the address of a machine-coded function.

User Response: Change the expression.

CCN6645 The bit-field ″%1$s″ is too small: %2$s bits are needed for ″%3$s″.

Where: ″%1$s″ is the bit-field. %2$s is the number of bits. ″%3$s″ is the name of the enumerated type.

Explanation: The size of the bit-field is not large enough to contain all of the possible values.

User Response: Increase the size of the bit-field.

CCN6646 The explicit instantiation of member ″%1$s″ must have a definition.

Where: ″%1$s″ is the member.

Explanation: The definition must be available in order for an instantiation to be done.

User Response: Define the static member.

CCN6647 The sizes of the pointer types of the argv parameter of function main are different.

Explanation: The sizes of the pointer types must match. This is an OS/400 (iSeries) message.

User Response: Use one of these, or an equivalent for the type of the argv parameter on main: char*__ptr128*__ptr128 char*__ptr64*__ptr64 Combinations such as the following are not allowed: char*__ptr64*__ptr128 char*__ptr128*__ptr64

CCN6800 The divisor for the modulus or division operator must not be zero.

Explanation: A division-by-zero condition has been detected.

User Response: Change the expression.

CCN6801 The result of expression evaluation resulted in an overflow.

Explanation: An overflow condition has been detected.

User Response: Change the expression.

CCN6802 The result of expression evaluation resulted in an underflow.

Explanation: An underflow condition has been detected.

User Response: Change the expression.

CCN8100 ″%1$s″ specified in ″%2$s″ is not a valid numeric value. The option is ignored.

Where: ″%1$s″ is the invalid numeric value. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because the argument was not a valid numeric value.

User Response: Verify the syntax of the option.

CCN8101 The numeric value ″%1$s″ specified in ″%2$s″ is out of bounds. The option is ignored.

Where: ″%1$s″ is the out-of-bounds value specified. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because the argument was not a numeric value within the range specified by this option.

User Response: Verify the allowable values for this option.

CCN8102 The alignment value ″%1$s″ specified in ″%2$s″ is not a power of two. The option is ignored.

Where: ″%1$s″ is the invalid alignment value. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because the alignment specified was not a power of two.

User Response: Verify the allowable values for this option.

CCN8103 ″%1$s″ specified in ″%2$s″ is not recognized. The option is ignored.

Where: ″%1$s″ is the unrecognized argument. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because the specified argument was not recognized.

User Response: Verify the syntax of the option.

CCN8104 The message number %1$s specified in ″%2$s″ is not a valid message ID. The option is ignored.

Where: ″%1$s″ is the invalid message ID. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because the message ID is not valid.

User Response: Verify the syntax of the option and the message ID.

CCN8105 A non-empty string is required but ″%1$s″ appears in ″%2$s″. The option is ignored.

Where: ″%1$s″ is the invalid argument. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because it was expecting a string with characters in it.

User Response: Verify the syntax of the option.

CCN8106 An option argument is required but is not found in ″%2$s″. The option is ignored.%1$s

Where: ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because it expected an argument which was not provided.

User Response: Verify the syntax of the option.

CCN8107 ″%1$s″ specified in ″%2$s″ contains embedded spaces. The option is ignored.

Where: ″%1$s″ is the argument containing embedded spaces. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored due to embedded spaces in the argument.

User Response: Verify the syntax of the option and the value passed as an argument.

CCN8108 The option argument ″%1$s″ specified in ″%2$s″ is not valid. The option is ignored.

Where: ″%1$s″ is the invalid argument. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because the argument specified was not valid.

User Response: Verify the syntax of the option.

CCN8109 The section attributes ″%1$s″ specified in ″%2$s″ are not valid. The option is ignored.

Where: ″%1$s″ is the invalid section attributes argument. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because the section attributes argument was not valid.

User Response: Verify the syntax of the option.

CCN8110 An unnecessary argument ″%1$s″ is found in ″%2$s″. The option is ignored.

Where: ″%1$s″ is the unnecessary argument. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because an unnecessary argument was specified.

User Response: Verify the syntax of the option.

CCN8111 ″%1$s″ specified in ″%2$s″ requires an additional option argument. The option is ignored.

Where: ″%1$s″ is the argument that requires more information. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because the argument requires more information.

User Response: Verify the syntax of the option.

CCN8120 The AlignAddr value ″%1$s″ is less than the AlignFile value ″%2$s″.

Where: ″%1$s″ is the AlignAddr value. ″%2$s″ is the AlignFile value.

Explanation: The AlignAddr value must be greater than the AlignFile value.

User Response: Change the values.

CCN8121 ″%1$s″ in ″%2$s″ is not a valid object model name. The option is ignored.

Where: ″%1$s″ is the invalid object model name specified. ″%2$s″ is the option being ignored.

Explanation: The option specified was ignored because the specified object model name was not valid.

User Response: Verify the option syntax.

CCN8122 ″%1$s″ is in conflict with ″%2$s″. The option is ignored.

Where: ″%1$s″ and ″%2$s″ are the conflicting options.

Explanation: The options specified are not valid if they are specified together.

User Response: Verify the options and remove or modify one of them.

CCN8123 The string ″%1$s″ in ″%2$s″ is not a valid identifier. The option is ignored.

Where: ″%1$s″ is the invalid identifier specified. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because it expected a valid identifier.

User Response: Verify the syntax of the option and the string specified.

CCN8124 The string ″%1$s″ in ″%2$s″ is not a valid keyword. The option is ignored.

Where: ″%1$s″ is the invalid string specified. ″%2$s″ is the option being ignored.

Explanation: The specified option was ignored because it expected a string containing a valid keyword.

User Response: Verify the syntax of the option and the string specified.

CCN8125 The option argument ″%1$s″ specified in ″%2$s″ is longer than %3$s characters. The option is ignored.

Where: ″%1$s″ is the invalid argument. ″%2$s″ is the maximum length. ″%3$s″ is the option being ignored.

Explanation: The specified option was ignored because the argument was too long.

User Response: Verify the syntax and constraints of the option.

CCN8130 The value ″%1$s″ in option ″%2$s″ is reserved for system use. The value is not accepted.

Where: ″%1$s″ is the value. ″%2$s″ is the option.

Explanation: The specified value is not accepted because it is reserved by the system.

User Response: Change the specified value.

CCN8131 The global option directive ″%1$s″ must not be placed inside braces. The option is ignored.

Where: ″%1$s″ is the option directive being ignored.

Explanation: The specified option directive is a global directive that applies to the target rather than to individual files.

User Response: Move the option to the global scope.

CCN8132 The global option directive ″%1$s″ is not allowed because it modifies a previous directive. The option is ignored.

Where: ″%1$s″ is the global option directive being ignored.

Explanation: The specified option directive is ignored because it conflicts with a previous directive.

User Response: Verify the meaning of the option directives specified to see that they do not conflict.

CCN8133 No include path is specified for the option ″%1$s″. The option is ignored.

Where: ″%1$s″ is the option being ignored.

Explanation: The specified option was ignored because it expected an include path as a an argument.

User Response: Verify the syntax of the option.

CCN8134 Error in setting option ″%1$s″ for extension source ″%2$s″. Configuration value ″%3$s″ has the wrong format.

Where: ″%1$s″ is the option. ″%2$s″ is the extension source. ″%3$s″ is the configuration value.

Explanation: This is a warning messsage about compiler extension source options.

User Response: If you are using that extension, use the correect option for that extension.

CCN8135 Default value of option ″%1$s″ in the .ice file has the wrong format ″%2$s″.

Where: ″%1$s″ is the option which has an invalid default value in the .ice file.

Explanation: The .ice file contains an invalid default value for the specified option.

User Response: Verify the syntax used to specify defaults in the .ice file.

CCN8136 Options ″%1$s″ and ″%2$s″ are in conflict.

Where: ″%1$s″ and ″$2$s″ are the conflicting options.

Explanation: The specified options cannot be specified together because they conflict.

User Response: Verify the option settings and remove or modify one of the conflicting options.

CCN8137 OBJECT_MODE setting ″%1$s″ is not recognized and is not a valid setting for the compiler.

Where: ″%1$s″ is the invalid setting.

Explanation: The specified OBJECT_MODE setting is not valid.

User Response: Verify the valid settings for OBJECT_MODE.

CCN8138 OBJECT_MODE = 32_64 is not a valid setting for the compiler.

Explanation: The 32_64 OBJECT_MODE setting is not supported.

User Response: Verify the valid settings for OBJECT_MODE.

CCN8139 The global option ″%1$s″ should be applied to all sources and targets.

Where: ″%1$s″ is the global option.

Explanation: A global option is an option that applies to all sources and targets rather than just one specified source file.

User Response: Move the global option so that it applies to all targets and sources.

CCN8140 ″%1$s″ is not compatible with 64-bit object mode. The default value ″%2$s″ is being set.

Where: ″%1$s″ is the option that is not valid for 64-bit object mode. ″%2$s″ is the default value being set.

Explanation: The specified option is not valid for 64-bit object mode, so the specified default is being set.

User Response: Verify the options that are valid for 64-bit object mode or switch to 32-bit object mode.

CCN8141 ″%1$s″ is not compatible with 32-bit object mode. The default value ″%2$s″ is being set.

Where: ″%1$s″ is the option which is not valid for 32-bit object mode. ″%2$s″ is the default value being set.

Explanation: The specified option is not valid for 32-bit object mode so the specified default is being set.

User Response: Verify the options that are valid for 32-bit object mode or switch to 64-bit object mode.

CCN8142 ″%1$s″ is not compatible with ″%2$s″. ″%3$s″ is being set.

Where: ″%1$s″ and ″%2$s″ are the incompatible option values. ″%3$s″ is the setting chosen by the compiler.

Explanation: The specified option values cannot be specified together because they are not compatible. A valid option is being set instead.

User Response: Verify the option values, and either remove or modify them so that they are compatible.

CCN8143 ″%1$s″ option is specified, but no floating point traps are being detected.

Where: ″%1$s″ is the option.

Explanation: Floating point traps are enabled but no traps have been specified.

User Response: Remove the option.

CCN8200 Class ″%1$s″ has base classes with different object models.

Where: ″%1$s″ is the name of the derived class.

Explanation: The object model deals primarily with the layout of class hierarchies. All classes in the same inheritance hierarchy must have the same object model.

User Response: Modify either the base class or the derived class so that both have the same object model.

CCN8201 Class ″%1$s″ is specified with a different object model than its base classes. The object model specified in its base classes will be used.

Where: ″%1$s″ is the name of the derived class.

Explanation: The object model deals primarily with the layout of class hierarchies. All classes in the same inheritance hierarchy must have the same object model.

User Response: Modify either the base class or the derived class so that both have the same object model.

CCN8202 Class ″%1$s″ has different object model between its formal template class and its base classes.

Where: ″%1$s″ is the name of the instance class.

Explanation: The object model deals primarily with the layout of class hierarchies. All classes in the same inheritance hierarchy must have the same object model.

Any formal templates (primary templates or partial specializations) must also have the same object model.

User Response: Modify either the base class or the formal template class so that both have the same object model.

CCN8400 ″%1$s″ is undefined. The delete operator will not call a destructor.

Where: ″%1$s″ is the class.

Explanation: The class is declared but not defined so a constructor will not be called when the object is deleted at this point.

User Response: Define the class.

CCN8401 The address of the destructor ″%1$s″ cannot be taken.

Where: ″%1$s″ is the destructor.

Explanation: An attempt has been made to take the address of a destructor.

User Response: Change the code to not take the address of the destructor.

CCN8402 The explicit reference to the destructor ″%1$s″ can only be used in an explicit destructor call.

Where: ″%1$s″ is the destructor.

Explanation: Destructors do not have names and can only be referred to in declarations and in pseudo-destructor calls.

User Response: Remove the reference to the destructor.

CCN8403 An expression with type pointer-to-member function must be bound to an object or a pointer to an object when it is used with the function call operator ().

Explanation: A pointer-to-member function must have an object to refer to when calling the function.

User Response: Change the code so that the function is being called on an object or a pointer to an object.

CCN8404 All the arguments must be specified for ″%1$s″ because its default arguments have not been checked yet.

Where: ″%1$s″ is the function.

Explanation: The function is recursive and is using the default arguments. Because they have not been processed yet, they must be specified.

User Response: Specify the parameters to the function call.

CCN8405 An empty initializer list cannot be used to initialize an unbounded array.

Explanation: The array is unbounded and its size is not known so an empty initializer list cannot be used.

User Response: Specify the size of the array or use a non-empty initializer list.

CCN8406 Build with the ″%1$s″ compiler option to extend the scope of the for-init-statement declaration.

Where: ″%1$s″ is the compiler option that can extend the scope of the variables declared in the for statement.

Explanation: Informational message about the option for extending scope of the variable in the for statement.

User Response: See the primary message.

CCN8407 The local macro ″%1$s″ is not visible in the current source.

Where: ″%1$s″ is the macro.

Explanation: Informational message about a local macro.

User Response: See the primary message.

CCN8408 The condition declaration cannot have type ″%1$s″.

Where: ″%1$s″ is the type.

Explanation: The type of the variable declared in the condition is not valid.

User Response: Change the type of the declaration in the condition to bool.

CCN8409 The condition declaration cannot be initialized with a brace list initializer.

Explanation: A declaration in a condition cannot be initialized with a brace list.

User Response: Change the initializer so that it is not in brace list format.

CCN8410 The left side of the ″%1$s″ operator must be an lvalue.

Where: ″%1$s″ is the operator.

Explanation: The operand on the left side is not an object that can be assigned a value.

User Response: Change the left operand to an object that can be assigned a value.

CCN8411 A dynamic cast is present, but the correct RTTI option is not specified.

Explanation: The compilation unit must be compiled with RTTI enabled.

User Response: Use the correct RTTI compiler option, or remove the dynamic cast.

CCN8412 A typeid is present, but the correct RTTI option is not specified.

Explanation: The compilation unit must be compiled with RTTI enabled.

User Response: Use the correct RTTI compiler option, or remove the type ID.

CCN8413 The ″__alignof″ operator cannot be applied to a bit-field.

Explanation: An attempt to use the __alignof operator on a bit-field has been made.

User Response: Remove the use of the __alignof operator.

CCN8600 ″%1$s″ operator cannot be overloaded.

Where: ″%1$s″ is the operator.

Explanation: The attempted operator overload is not valid.

User Response: Change the declaration to overload a different operator.

CCN8601 Forward declaration of the enumeration ″%1$s″ is not allowed.

Where: ″%1$s″ is the enumeration.

Explanation: Enumerations cannot have forward declarations.

User Response: Define the enumeration before attempting to use an elaboration of the enumeration.

CCN8602 The first non-matching token was encountered on line %1$s, column %2$s. A project cannot contain more than one definition of a class unless each definition consists of the same sequence of tokens.

Where: ″%1$s″ is the line number. ″%2$s″ is the column number.

Explanation: Informational message indicating the first token that differs in the two class definitions.

User Response: See the primary message.

CCN8701 The ″#pragma datamodel″ stack is empty. The pragma is ignored.

Explanation: An attempt has been made to restore the previous pragma setting, but this is the first instance of the pragma.

User Response: Remove the pragma.

CCN8702 Invalid syntax for #pragma datamodel.

Explanation: The compiler has detected an invalid #pragma datamodel syntax.

User Response: Correct the syntax.

CCN8703 #pragma datamodel(LLP64 | P128) seen without matching #pragma datamodel(pop).

Explanation: At the end of compilation there was an extra #pragma datamodel on the stack.

User Response: Ensure that all #pragma datamodel directives have a matching #pragma datamodel(pop).

CCN8704 The base class has a different data model than this derived class.

Explanation: Base and derived classes must have identical data models.

User Response: Change the data model of one of the classes.

CCN8705 Cannot initialize a static __ptr64 with a __ptr128 value.

Explanation: A __ptr64 variable is being initialized with a constant value when the storage model indicates such values are __ptr128s.

User Response: Use a different initialization value or a different storage model.

CCN8706 This message is no longer used.

CCN8707 The #pragma map has been applied to function ″%1$s″, which has internal linkage.

Where: %1$s is the function name.

Explanation: An internal linkage function cannot be mapped.

User Response: Change the function or remove the pragma.

CCN8708 **The divisor for the modulus or division operator cannot be zero.**

Explanation: The result of the calculation is undefined.

User Response: Change the value of the divisor or change the operator.

CCN8709 **The #pragma ″%1$s″ directive must occur before the first C++ statement in program; The directive is ignored.**

Where: %1$s is the pragma name.

Explanation: The pragma must precede any C++ statement in the program.

User Response: Move the pragma directive before any C++ statement.

Note: The following error messages may be produced by the compiler if the message file is itself invalid.

SEVERE ERROR EDC0090: Unable to open message file *&1*.
SEVERE ERROR EDC0091: Invalid offset table in message file *&1*.
SEVERE ERROR EDC0092: Message component *&1s* not found.
SEVERE ERROR EDC0093: Message file *&1* corrupted.
SEVERE ERROR EDC0094: Integrity check failure on msg *&1*
SEVERE ERROR EDC0095: Bad substitution number in message *&1*
SEVERE ERROR EDC0096: Virtual storage exceeded
ERROR: Failed to open message file. Reason *&1*.
ERROR: Unable to read message file. Reason *&1*.
ERROR: Invalid offset table in message file *&1*.
ERROR: Message component *&1s* not found.
ERROR: Message file *&1* corrupted.
ERROR: Integrity check failure on msg *&1* — retrieved *&2*.
ERROR: Message retrieval disabled. Cannot retrieve *&1*.
INTERNAL ERROR: Bad substitution number in message *&1*.

Note: The previous messages are only generated in English.

Chapter 4. Utility Messages

This chapter contains information about the `DSECT`, `DLLRNAME`, and `CXXFILT` utility messages, and should not be used as programming interface information. For the `localedef`, `iconv`, and `genxlt` utility messages, refer to the *z/OS Language Environment Debugging Guide*.

Other Return Codes and Messages

See the *z/OS Language Environment Debugging Guide* for messages and return codes for the following:

- Prelinker and Object Library Utility
- Run-time messages and return codes
- localedef utility
- genxlt utility
- iconv utility
- System Programmer C (SP C)

DSECT Utility Messages

The following section describes return codes and messages that are issued by the DSECT utility.

Return Codes

The DSECT utility issue the following return codes:

Table 4. Return Codes from the DSECT Utility

Return Code	Meaning
0	Successful completion.
4	Successful completion, warnings issued.
8	DSECT Utility failed, error messages issued.
12	DSECT Utility failed, severe error messages issued.
16	DSECT Utility failed, insufficient storage to continue processing.

Messages

The messages that the DSECT utility issues have the following format:

EDCnnnns text <*s*> where:

nnnn	error message number
s	error severity
00	informational message
10	warning message
30	error message
40	severe error message
&s	substitution variable

The DSECT utility issues the following messages:

EDC5500 10 Option %s is not valid and is ignored.

Explanation: The option specified in the message is not valid DSECT Utility option or a valid option has been specified with an invalid value. The specified option is ignored.

User Response: Rerun the DSECT Utility with the correct option.

EDC5501 30 No DSECT or CSECT names were found in the SYSADATA file.

Explanation: The SECT option was not specified or SECT(ALL) was specified. The SYSADATA was searched for all DSECTs and CSECTs but no DSECTs or CSECTs were found.

User Response: Rerun the DSECT Utility with a SYSADATA file that contains the required DSECT or CSECT definition.

EDC5502 30 Sub option %s for option %s is too long.

Explanation: The sub option specified for the option was too long and is ignored.

EDC5503 30 Section name %s was not found in SYSADATA File.

Explanation: The section name specified with the SECT option was not found in the External Symbol records in the SYSADATA file. The C structure is not produced.

User Response: Rerun the DSECT Utility with a SYSADATA file that contains the required DSECT or CSECT definition.

EDC5504 30 Section name %s is not a DSECT or CSECT.

Explanation: The section name specified with the SECT option is not a DSECT or CSECT. Only a DSECT or CSECT names may be specified. The C structure is not produced.

EDC5505 00 No fields were found for section %s, structure is not produced.

Explanation: No field records were found in the SYSADATA file that matched the ESDID of the specified section name. The C structure is not produced.

EDC5506 30 Record length for file ″%s″ is too small for the SEQUENCE option, option ignored.

Explanation: The record length for the output file specified is too small to enable the SEQUENCE option to generate the sequence number in columns 73 to 80. The available record length must be greater than or equal to 80 characters. The SEQUENCE option is ignored.

EDC5507 40 Insufficient storage to continue processing.

Explanation: No further storage was available to continue processing.

User Response: Rerun the DSECT Utility with a larger region (MVS).

EDC5508 30 Open failed for file ″%s″: %s

Explanation: This message is issued if the open fails for any file required by the DSECT Utility. The file name passed to fopen() and the error message returned by strerror(errno) is included in the message.

User Response: The message text indicates the cause of the error. If the file name was specified incorrectly on the OUTPUT option, rerun the DSECT Utility with the correct file name.

EDC5509 40 %s failed for file ″%s″: %s

Explanation: This message is issued if any error occurs reading, writing or positioning on any file by the DSECT Utility. The name of the function that failed (Read, Write, fgetpos, fsetpos), file name and text from strerror(errno) is included in the message.

User Response: This message may be issued if an error occurs reading or writing to a file. This may be caused by an error within the file, such as an I/O error or insufficient disk space. Correct the error and rerun the DSECT Utility.

EDC5510 40 Internal Logic error in function %s

Explanation: The DSECT Utility has detected that an error has occurred while generating the C structure. Processing is terminated and the C structure is not produced.

User Response: This may be caused by an error in the DSECT Utility or by incorrect input in the SYSADATA file. Contact your system administrator.

EDC5511 10 No matching right parenthesis for %s option.

Explanation: The option specified had a sub option beginning with a left parenthesis but no right parenthesis was present.

User Response: Rerun the DSECT Utility with the parenthesis for the option correctly paired.

EDC5512 10 No matching quote for %s option.

Explanation: The OUTPUT option has a sub option beginning with a single quote but no matching quote was found.

User Response: Rerun the DSECT Utility with the quotes for the option correctly paired.

EDC5513 10 Record length too small for file ″%s″.

Explanation: The record length for the Output file specified is less than 10 characters in length. The minimum available record length must be at least 10 characters.

User Response: Rerun the DSECT Utility with an output file with a available record length of at least 10 characters.

EDC5514 30 Too many sub options were specified for option %s.

Explanation: More than the maximum number of sub options were specified for the particular option. The extra sub options are ignored.

EDC5515 00 HDRSKIP option value greater than length for section %s, structure is not produced.

Explanation: The value specified for the HDRSKIP option was greater than the length of the section. A structure was not produced for the specified section.

User Response: Rerun the DSECT Utility with a smaller value for the HDRSKIP option.

EDC5516 10 SECT and OPTFILE options are mutually exclusive, OPTFILE option is ignored

Explanation: Both the SECT and OPTFILE options were specified, but the options are mutually exclusive.

User Response: Rerun the DSECT Utility with either the SECT or OPTFILE option.

EDC5517 10 Line %i from ″%s″ does not begin with SECT option

Explanation: The line from the file specified on the OPTFILE option did not begin with the SECT option. The line was ignored.

User Response: Rerun the DSECT Utility without OPTFILE option, or correct the line in the input file.

EDC5518 10 setlocale() failed for locale name ″%s″.

Explanation: The setlocale() function failed with the locale name specified on the LOCALE option. The LOCALE option was ignored.

User Response: Rerun the DSECT Utility without LOCALE option, or correct the locale name specified with the LOCALE option.

DLLRNAME Utility Messages

Return Codes

The `DLLRNAME` utility returns the following return codes:

Table 5. Return Codes from the DLLRNAME Utility

Return Code	Meaning
0	Processing successful.
8	Invalid input arguments
16	Any other failure

Messages

The DLLRNAME utility issues the following messages:

EDC6200E An invalid argument list was specified.

Explanation: The parameter list specified is not valid. See the documentation for DLLRNAME in the *z/OS C/C++ User's Guide*.

User Response: Ensure that you have included at least one application load module or DLL and that you have specified the options correctly and with the correct syntax.

EDC6201S A failure occurred accessing &.

Explanation: An unexpected error occurred when DLLRNAME tried to access the input file.

User Response: Look up the subsequent perror() message and perform the Programmer Response. For example, a `file not found` error may indicate that you need to fix the input file name. Otherwise, report the problem to IBM Service.

EDC6202S A DLL name & is already imported

Explanation: You have specified a DLL to rename. The new name chosen matches a DLL already in the import list.

User Response: Either change the new name to a name not already imported or first rename the DLL that has the chosen name.

EDC6203E A DLL name was specified more than once for a rename

Explanation: You have specified a DLL more than once in the *oldname=newname* list. The following are examples of invalid input: `A=B A=C` or `A=B B=C` or `A=A` or `A=C B=C`.

User Response: Fix the argument list so that the DLL appears only once.

CXXFILT Utility Messages

Return Codes

The `CXXFILT` utility returns the following return codes:

Table 6. Return Codes from the CXXFILT Utility

Return Code	Meaning
0	Processing successful: CXXFILT processing completed successfully.
4	A warning was issued and a result was generated.
8	CXXFILT Utility failed, possibly due to a read error.
16	CXXFILT Utility failed.

Messages

The CXXFILT utility issues the following messages:

CCN9500 Cannot open the following file: @1 -- ignored.

Explanation: The specified file cannot be opened for reading or does not exist.

User Response: Ensure that the file exists and is readable.

CCN9501 Cannot continue reading input.

Explanation: A read error occurred while reading the input stream.

User Response: Ensure that the input stream is still available and try again.

CCN9502 No options specified after (.

Explanation: A (indicating start of options was encountered but no options followed.

User Response: Ensure that the input stream is still available and try again.

CCN9503 An invalid option (@1) was specified -- ignored.

Explanation: An invalid option was specified.

User Response: Refer to the OS/390 C/C++ User's Guide under cxxfilt for valid options.

CCN9504 Option (@1) was specified with too few suboptions. @2 suboption(s) required -- ignored.

Explanation: Not all the required suboptions were supplied.

User Response: Refer to the OS/390 C/C++ User's Guide under cxxfilt for the number of required suboptions.

CCN9505 Option (@1) was specified with too many suboptions. @2 suboption(s) required -- ignored.

Explanation: More suboptions were supplied than what is allowed by this option.

User Response: Refer to the OS/390 C/C++ User's Guide under cxxfilt for the number of required suboptions.

CCN9506 Option (@1) requires a positive suboption -- ignored.

Explanation: This error occurred because the specified suboptions for this option is invalid. Only positive suboptions are allowed.

User Response: Refer to the OS/390 C/C++ User's Guide under cxxfilt for the allowed suboptions.

CCN9507 Internal Error. Contact your IBM representative.

User Response: Please report this problem.

CCN9508 No negative form for option @1 -- ignored.

Explanation: The specified option does not have a negative form.

User Response: Refer to the OS/390 C/C++ User's Guide under cxxfilt for valid options.

CCN9509 An incomplete option (@1) has been specified. -- ignored

Explanation: The specified option is incomplete.

User Response: Refer to the OS/390 C/C++ User's Guide under cxxfilt for valid options.

Chapter 5. z/OS C/C++ Application Support Class Library and Collection Class Library Version 5 Messages

This chapter contains information about the Version 5 Application Support Library and Collection Class Library messages that are included with z/OS V1R2 IBM Open Class and should not be used as programming interface information.

The following information shows the format for these messages:

Message Format: **CLEnnnn text <&*n*>** where:

nnnn	error message number
text	message which appears on the screen
&*n*	IBM Open Class substitution variable

CLE1000 **The ″&1″ expression must be true, but it evaluated to false.**

Where: &1 is the expression

User Response: Check the variables in the expression.

CLE1001 **Out of memory or unable to allocate memory.**

Explanation: The attempt to obtain memory in order to satisfy the current library request has failed.

User Response: Make sure that there is enough memory in the region to run the program. You might need to specify the run-time option HEAP(,,,FREE,,) to prevent the program from running out of memory.

CLE1002 **The class or the called member function is not supported.**

Explanation: The IInvalidRequest exception is thrown because the user application called a member function of the class or tried to instantiate an instance of a class which is supported only in an z/OS or OS/390 UNIX Environment.

User Response: Change your application logic to avoid calling the member function or creating an instance of class which is not supported in a z/OS or OS/390 non-UNIX Environment.

CLE1003 **A system exception condition was detected.**

Explanation: The ISystemErrorInfo exception is thrown to indicate that an operating system call resulted in an error condition. Typically, this exception is thrown in IDLLModule routines, for example, IDLLModule::close().

User Response: Check for IDLLModule operations used in the application.

CLE1004 **A protected function of a class was called; it can produce unpredictable results.**

Explanation: A user application called a protected function of a class. This can result in unpredictable behavior.

User Response: Remove the call to the protected function of the class.

CLE1100 **The file system operation was cancelled.**

Explanation: A user application cancelled a file system operation through the IFileOperationCancelled exception or this exception is thrown when the reportProgress() function in the IFileOperation subclass is false or when a file system operation is interrupted by a signal.

User Response: If you intended to stop the operation, no response is needed. Otherwise, note the location of the problem and contact IBM C++ Service and Support.

CLE1101 **This operating system does not support this function.**

Explanation: A user application has called a function that is not supported on this operating system.

User Response: Do not use the function in question.

CLE1102 **A file system volume cannot be deleted.**

Explanation: The IInvalidRequest exception is thrown because you tried to delete or move a volume.

User Response: Do not use the deleteSelf or moveTo functions on volumes.

CLE1103 A file system volume cannot be renamed.

Explanation: The IInvalidRequest exception is thrown because you tried to rename a volume.

User Response: Do not use the setName function to rename volumes.

CLE1104 The file system object is invalid.

Explanation: The IEntityInvalid exception is thrown because of an attempt to perform an operation in an IFileSystemEntity object which does not point to a valid on-disk entity.

User Response: Find out the location of the problem and remove reference to the invalid file system entity.

CLE1105 The file system entity was not found.

Explanation: A user application has referenced a file system entity which was not found.

User Response: Find out the location of the problem and remove reference to the non-existent file system entity.

CLE1106 The file system entity already exists.

Explanation: A user application has attempted to create a file system entity that already exists.

User Response: Find out the location of the problem and remove reference to the file system entity in question.

CLE1107 The volume is offline.

Explanation: The IVolumeOffline exception is thrown because of an attempt to operate on an offline or ejected volume.

User Response: Find out the location of the problem and remove reference to the offline volume.

CLE1108 The file name is invalid.

Explanation: The IInvalidParameter exception is thrown because the path name specified by user application is invalid.

User Response: Find out the location of the problem and remove the reference to the invalid file name.

CLE1109 The directory is not empty.

Explanation: The directory path name specified by user application is not empty.

User Response: Make sure that the directory path name is empty before it is deleted.

CLE1110 The file system is in use.

Explanation: The IEntityInUse exception is thrown because an attempt has been made to move, delete or operate on an entity which is currently in use.

User Response: Make sure that the file system entity is not used before an operation is performed on it.

CLE1111 Attempted to read beyond the end of file.

Explanation: The end of the file has been encountered prematurely.

User Response: Make sure that file position offset used does not reach beyond the end of file.

CLE1112 The file system entity is not the correct type.

Explanation: The IEntityTypeMismatch exception indicates an attempted assignment to the wrong type of entity object by either a user application or typeSafeAssign member function in the IFileSystemEntity subclass.

User Response: If you intended to use this exception, no response is needed. Otherwise, check that the correct type of a file system entity is used in any of the following: typeSafeAssign member function, IFileSystemEntity subclass constructor, or IFileSystemEntity = operator.

CLE1113 &1 is not a file.

Explanation: The IEntityTypeMismatch exception indicates an attempted assignment to the wrong type of a file by either user application or typeSafeAssign member function in the IFileSystemEntity subclass.

User Response: If you intended to use this exception, no response is needed. Otherwise, check that the correct type of a file is used in any of the following: typeSafeAssign member function, IFile constructor, or IFile = operator.

CLE1114 &1 is not a directory.

Explanation: The IEntityTypeMismatch exception indicates that an attempted assignment to the wrong type of directory was made by either the user application or the typeSafeAssign member function in the IFileSystemEntity subclass.

User Response: If you intended to use this exception, no response is needed. Otherwise, check that the correct type of a directory is used in the typeSafeAssign member function, IDirectory constructor, or IDirectory = operator.

CLE1115 &1 is not a volume.

Explanation: The IEntityTypeMismatch exception indicates that an attempted assignment to the wrong type of volume was made by either the user application or the typeSafeAssign member function in the IFileSystemEntity subclass.

User Response: If you intended to use this exception, no response is needed. Otherwise, check that the correct type of volume is used in the typeSafeAssign member function, IVolume constructor, or IVolume = operator.

CLE1116 The ″&1″ value is not supported on this platform.

Explanation: The IInvalidRequest exception is thrown because the value used in the parseName() routine is not recognized on this platform.

User Response: Make sure that you are using the correct parameter in the parseName() routine.

CLE1117 Object ″&1″ cannot be found.

Explanation: An exception is thrown because the value used in the parseName() routine is not recognized as a valid name.

User Response: Make sure that you are using the correct parameter in the parseName() routine.

CLE1200 The DBCS string is invalid because a shift-out or shift-in character is missing.

Explanation: DBCS characters in the MBCS string are not enclosed in shift-out and shift-in characters. Either a shift-out or a shift-in character is missing.

User Response: Enclose DBCS characters in an MBCS string in shift-out and shift-in characters.

CLE1201 An error occurred while converting an MBCS string to a wide character string.

Explanation: DBCS characters in the MBCS string are not enclosed in shift-out and shift-in characters. Either a shift-out or a shift-in character is missing.

User Response: Enclose DBCS characters in an MBCS string in shift-out and shift-in characters.

CLE1202 Data overflow error.

Explanation: A decimal data overflow was detected and the IDecimalDataError exception was thrown. Overflow errors can occur while constructing an IBinaryCodedDecimal or IDecimalUtil object from a number or while performing arithmetic operations on these objects.

User Response: Change the decimal constant used in the IBinaryCodedDecimal or IDecimalUtil class so that it does not exceed the maximum value.

CLE1204 Attempt to allocate an IString that is too big.

Explanation: The IInvalidRequest exception indicates that an IString request caused an overflow. An exception is thrown when IBuffer::checkAddition or IBuffer::checkMultiplication detects an overflow. Typically, this occurs during object construction or during an operation that grows an underlying IBuffer object.

User Response: Check the usage of IString objects in your application and make sure that they are within acceptable range.

CLE1205 Attempt to index beyond the end of a const IString.

Explanation: The IInvalidRequest exception was thrown because the value of the IString designator was larger than the dimension declared for an IString array.

User Response: Increase the size of the IString array. Make sure you include space for the terminating null character.

CLE1206 Text offset out of bounds.

Explanation: The IInvalidParameter exception is thrown because the offset of the next text boundary exceeded the length of the text.

User Response: Make sure that the offset where scanning begins, which is specified as a parameter, does not exceed the length of the text. Check the usage of the ITextBoundary class.

CLE1207 State not initialized.

Explanation: The IInvalidParameter exception is thrown because the internal Transcoding state has not been initialized.

User Response: Make sure that the conversion functions which reset internal Transcoding states are used correctly.

CLE1208 State not initialized or buffer range error.

Explanation: The IInvalidParameter exception is thrown because either an internal Transcoding state has not been initialized or the string buffer range is incorrect.

User Response: Make sure that conversion functions have correct parameters.

CLE1300 Unsupported member function of IThread class called.

Explanation: The IInvalidRequest exception is thrown because a user application has called a member function of the IThread class which is not supported on this platform.

User Response: Change your application logic to avoid calling this member function.

CLE1301 The specified thread ID is not valid.

Explanation: The IInvalidParameter exception is thrown because a user application passed an invalid thread ID to the IThread class.

User Response: Ensure that a thread ID that is valid for started threads has been passed to IThread class.

CLE1302 The start() function is not valid because the specified thread is already started.

Explanation: The IInvalidRequest exception is thrown because a user application called the start() function on the IThread class, but the thread was already running.

User Response: Check your application to ensure that the start() function is called after the previous function dispatched on the IThread has been completed.

CLE1303 Unable to acquire a semaphore to satisfy the setLock() request.

Explanation: The IOutOfSystemResource exception is thrown because no more semaphore resource is available to complete the user request. Most likely the system limit for the number of semaphores has been exceeded.

User Response: Free up any unused semaphore resources that your application acquired, and try the request again. If the problem persists, contact your system representative to free the unused semaphore resources.

CLE1304 An attempt was made to allocate a keyed thread variable beyond the library's limit of 16.

User Response: Check your application to ensure that the number of keyed thread variables is below the maximum limit.

CLE1305 The thread is not started.

Explanation: The IInvalidRequest exception is thrown if the thread is invalid or the thread has not been started. This typically occurs when INonGUIThread::resume() cannot resume executing a thread for the previously stated reasons.

User Response: Check that started threads have valid thread IDs and that threads in use have been started.

CLE1306 The thread is not in a suspended state.

Explanation: The IAccessError exception is thrown if the thread is invalid or not in the suspended state. This typically occurs when INonGUIThread::resume() cannot resume executing a thread for the previously stated reasons.

User Response: Check that started threads have valid thread IDs and that threads have been suspended before making a call to resume theme.

CLE1400 The data on the stream is not in the expected format.

Explanation: The IInvalidDataOnStream exception is thrown because invalid data was read from IDataStream.

User Response: Make sure that IDataStream contains valid data.

CLE1401 An attempt was made to read past the end of the stream.

Explanation: The IEndOfStream exception is thrown because reading was performed past the end of IDataStream.

User Response: Check for the end of IDataStream before reading IDataStream data.

CLE1402 An invalid stream encoding was specified.

Explanation: The IInvalidParameter exception is thrown because the incorrect type of stream encoding was used. This typically occurs when an IDataStream::createMemoryStream() member function uses an invalid type of stream encoding. Valid encoding types are: kInteroperableBinary, kRawBinary, and kDebug.

User Response: Make sure that valid stream encoding is specified.

CLE1403 An unknown type was encountered on the stream.

Explanation: The IUnknownTypeOnStream exception is thrown because an object with an unknown type was encountered while reading from IDataStream. One situation that can cause this error is that the dynamic link library, or shared library, with the class implementation for the type is not loaded in the process that is reading from the stream.

User Response: Make sure that valid source is streamed.

CLE1404 An object previously seen on the stream was reallocated.

Explanation: The IAddressAlreadyInContext exception is thrown because there is an internal inconsistency in the handling of aliased objects in the data stream. This problem could be caused by corruption of the stream data, programming errors in the application, or defects in the streaming classes. Typically, the exception is thrown while reading from or writing C++ objects to IDataStream.

User Response: Make sure that valid source is streamed, and that valid data is being read from or written to IDataStream.

CLE1405 The context number is out of range.

Explanation: The IInvalidContextNumber exception is thrown because an object alias read from IDataStream refers to data that does not exist. This is an internal consistency check; a failure generally indicates that the stream data has been corrupted, or that some other serious programming error has occurred.

User Response: Make sure that valid source is streamed, and that valid data is being read from or written to IDataStream.

CLE1500 Transcoding buffer range is invalid.

Explanation: The IInvalidParameter exception indicates that the string range is invalid. Typically, this exception appears in the ITranscoder::toUnicode() or ITranscoder::fromUnicode() member function to indicate that either a foreign code set or a Unicode parameter pointer is NULL, or that the pointer to end of the buffer range from_end or to_limit points to an invalid location, for example, from_end - from < 0.

User Response: Make sure that transcoders used contain valid transcoding buffer range.

CLE1501 Transcoding flush buffer is invalid.

Explanation: IInvalidParameter exception is thrown in ITranscoder::flush() member function to indicate that output conversion buffer points to either a NULL location or that the range of the output conversion buffer is invalid (e.g. (to_limit-to) < 0). Note: ITranscoder::flush() member function does nothing in ITranscoder base class, it simply checks for valid parameters.

User Response: Make sure that parameters in ITranscoder::flush() function are correct.

CLE1503 Transcoder for charset or locale specified is not installed.

Explanation: The IObjectNotFound exception indicates that the transcoder used either contains an invalid locale ID or that the character set for the given transcoder is invalid.

User Response: Check for valid transcoders specified in ITranscoder objects.

CLE1505 An input byte does not belong to the input codeset.

Explanation: The IInvalidRequest exception is thrown to indicate that an illegal character was used for a given codeset. This typically occurs when input buffer parameters in the ITranscoder::toUnicode() or ITranscoder::fromUnicode() member function contain invalid characters for the codesets used.

User Response: Make sure that the transcoding buffers contain valid characters for codesets used.

CLE1506 Incomplete character or shift sequence at the end of the input buffer.

Explanation: The IInvalidRequest exception is thrown to indicate that the input transcoding buffer in the ITranscoder::toUnicode() or ITranscoder::fromUnicode() member function ends with an incomplete character or shift sequence. Conversion stops after the previous successfully converted bytes and the exception is thrown.

User Response: Make sure that the transcoding buffers contain valid characters for the codesets used.

CLE1600 The object cannot be located.

Explanation: The IObjectNotFound exception indicates that an operation failed because it was unable to locate a requested object, for example, an invalid file system object, file or directory.

User Response: If you intended to use this exception, no response is required. Otherwise, check the usage of incorrect objects in your application.

CLE1601 The name is invalid in this context.

Explanation: The IInvalidName exception indicates that an invalid name, such as a file name or a network resource name, was used for an object.

User Response: If you intended to use this exception, no response is required. Otherwise, check for usage of proper names in your application.

CLE1602 The operation cannot proceed, but might be able to continue.

Explanation: The ICannotProceed exception indicates a situation prohibiting the application from completing an operation.

User Response: If you intended to use this exception, no response is required.

CLE1603 The object already exists.

Explanation: The IAlreadyExists exception indicates that an operation could not create the requested object because the name of the object or its location is already used. This exception is typically used in file system operations.

User Response: Check the object creation operations in your application and make sure that objects that you want to create do not already exist.

CLE1604 The container must be empty.

Explanation: The IMustBeEmpty exception was thrown to indicate that an operation could not be performed on a container because the container was not empty. This typically occurs when you are trying to remove a directory which is not empty.

User Response: Check for container operations used in the application.

CLE1800 Process Invalid path: no permission to execute.

Explanation: The IInvalidRequest exception indicates that either the external process you are trying to start does not have permission to execute or the directory of the process you are trying to start does not have permission to search.

User Response: Check that the IExternalProcess::start() function uses valid process locations.

CLE1801 Invalid process path: not a regular file.

Explanation: The IInvalidRequest exception indicates that the process you are trying to start is not a regular file.

User Response: Check that the IExternalProcess::start() function uses valid process locations.

CLE1802 Process ″&1″ already started, cannot start ″&2″.

Explanation: The IInvalidRequest exception indicates that a process cannot be started because it has already been started.

User Response: Check for already started processes that you are attempting to start in your application. Check the usage of IExternalProcess class.

CLE1803 Setting process path failed.

Explanation: The IInvalidRequest exception is thrown because an external process path was being set while the process was running. This typically occurs in a call using the IExternalProcess::start() or IExternalProcess::setPa th() functions to a process that is already running.

User Response: Make sure that you are not attempting to change the path name of a process that is already running.

CLE1804 Failed to set process environment.

Explanation: The IInvalidRequest exception indicates that an external process environment was being set while the process was running. This typically occurs in a call using the IExternalProcess::setEnvironment() function to a process that is already running.

User Response: Make sure that you are not attempting to change the environment of a process that is already running.

CLE1806 Cannot start the process because it is already running.

Explanation: The IInvalidRequest exception indicates that a process cannot be started because it has already been started. This typically occurs when trying to start the process again using the IExternalProcess::start() function.

User Response: Check for processes that are already started that you are attempting to start in your application with the IExternalProcess::start() function.

CLE1810 Unknown priority for the ″&1″ process.

Explanation: The IExternalProcess::getPriority() call returned a priority that is not mappable to a standard Open Class Library process priority class.

User Response: Avoid calling IExternalProcess::getPriority() for this process, because its priority is not recognized by the IBM Open Class Library.

CLE1811 Could not start the ″&1″ process. Error=&2.

Explanation: The IInvalidParameter exception indicates that an error occurred while trying to start a process. This happened in a IExternalProcess::start() call.

User Response: Check whether you are trying to start a valid process that has not already been started. Check that you are using the correct path name.

CLE1813 The ″&1″ operation is not supported on this platform.

Explanation: This exception typically occurs in IExternalProcess::setPriority() or IExternalProcess::getPriority() calls, which are not supported on certain platforms.

User Response: Change your application logic to avoid calling the unsupported function.

CLE1814 Error ″&1″ occurred while waiting on process ″&2″ to die.

Explanation: The IAccessError exception indicates that an error occurred while waiting for a process to die. This typically occurs in an IExternalProcess::wait() call.

User Response: Make sure that you are waiting for a valid process to die.

CLE1815 The program ″&1″ could not be located.

Explanation: The IInvalidParameter exception indicates that the external process you are trying to start was not found. This typically occurs in a call to the IExternalProcess::start() function.

User Response: Make sure that you are starting a valid external process with the correct path name. Ensure that your application has search permission on the process path name.

CLE1816 Path ″&1″ is not a valid host platform path.

Explanation: The IInvalidParameter exception indicates that the path name specified in the IExternalProcess::start() call is not a valid path name for an external process.

User Response: Make sure that the path name in the IExternalProcess::start() call is correct.

CLE2000 IThreadLocal Storage objects do not support re-adoption.

Explanation: The IInvalidRequest exception indicates that the calling thread has already stored an object, and this object does not allow resetting of the data value. The ability to reset is indicated during the construction of the IThreadLocalStorage object.

User Response: Make sure that you are not trying to repeatedly set the storage for a calling thread when the object does not allow resetting of the data value.

CLE2001 Invalid advancing request for locale key iterator.

Explanation: The IInvalidRequest exception indicates that the last locale key has been consumed and that advancing to the next locale key cannot be performed.

User Response: Avoid advancing the locale key past the limit when using a ++ operator in ILocaleKey.

CLE2002 The specified locale is not installed.

Explanation: The IInvalidRequest exception indicates that an error occurred while setting the application's locale. Failure to set the locale can be a result of an incorrect locale value or an incorrect category value.

User Response: Make sure that the locale is correct for strings used in ICollation functions or IDecimalFormat::format().

CLE2003 Invalid locale identifier.

Explanation: The IInvalidParameter exception indicates that the locale identifier used in the application is not valid for this operating system.

User Response: Make sure that locale identifiers used in ILocaleKey objects and operations are correct.

CLE2004 Locale supported but not installed.

Explanation: The IInvalidRequest exception indicates that the matched locale is supported but not installed on the machine. Typically occurs in calls to ILocaleKey constructor, ILocaleKey::setHostID() or ILocaleKey::setPOSIXID() functions.

User Response: Make sure that the locale identifiers specified in the functions listed above are correct.

CLE2005 Invalid POSIX identifier.

Explanation: The IInvalidParameter exception indicates that the POSIX locale identifier specified in ILocaleKey operations is incorrect.

User Response: Make sure that the locale identifier specified in ILocaleKey operations are correct.

CLE2009 Invalid look-up strategy.

Explanation: The IInvalidParameter exception is thrown to indicate that the look-up strategy parameter is invalid.

User Response: Check the lookup strategy value for a locale key display name parameter specified in displayName(). Valid lookup strategy values are: kExactLocale, kExactLanguage, or kAnyLanguage.

CLE2010 No locale path specified.

Explanation: The IInvalidRequest exception indicates that the LOCPATH environmental variable was not set.

User Response: Please specify the HFS directory from which to load locale object files in the LOCPATH environmental variable.

CLE2011 No locales installed.

Explanation: The IInvalidRequest exception indicates that the ILocaleKeyIterator was not instantiated using the POSIX interface and the LOCPATH environmental variable was not specified.

User Response: Make sure that the LOCPATH environmental variable points to the proper HFS directory from which to load locale object files.

CLE2100 Invalid advancing request for collation iterator.

Explanation: The IInvalidRequest exception indicates that the last collation object has been consumed and that advancing to the next collation table cannot be performed.

User Response: Avoid advancing collation iterator past the limit by using the ++ operator in ICollationIterator.

CLE2101 Incorrect text iterators.

Explanation: The IInvalidParameter exception is thrown because the provided text iterator whose offset is at the end of the string is greater than the text iterator whose offset is at the beginning of the string.

User Response: Check the text iterators provided in ICollation classes member functions such as compare(), hash() or transform() and make sure that the iterator whose offset is at the beginning of the string is less than the iterator whose offset is at the end of the string.

CLE2102 Incorrect text offsets.

Explanation: The IInvalidParameter exception indicates that the provided text offset for the end of the string is less than the text offset for the beginning of the string.

User Response: Check the text offsets provided in hash() or compare() functions of ICollation classes and make sure that the end of string offset is greater than the beginning of string offset.

CLE2103 Unknown system error occurs when calling wcsxfrm().

Explanation: The IInvalidRequest exception is thrown while transforming a wide character string.

User Response: Check for proper string usage in the ICollation::transform() function.

CLE2200 An attempt was made to dereference a null IReference.

Explanation: The IInvalidRequest exception was thrown because a null IReference pointer was used with a * operator or -> operator.

User Response: Check the usage of IReference pointers and make sure they are not null if dereferenced.

CLE3000 A child already exists.

Explanation: A child already exists at the given position.

User Response: Make sure there is no child at the position where you want to add one, for example, in routines such as addAsChild(), attachAsChild(), and attachSubtreeAsChild().

CLE3001 The collection is empty.

Explanation: The collection used in the application is empty.

User Response: Check your program to ensure that you added at least one element to the collection.

CLE3002 The cursor is not contained in this collection.

Explanation: The element corresponding to the cursor might have been removed from the collection.

User Response: Check your program to ensure that the cursor used in collection routines points to an element of the collection.

CLE3003 The cursor is not for the given collection.

Explanation: The cursor does not belong to the given collection.

User Response: Check your program to ensure that the cursor used in collection routines points to an element belonging to the given collection.

CLE3004 The cursor is not for this collection.

Explanation: The cursor does not belong to the collection to which the collection member function issuing this message is applied.

User Response: Check your program to ensure that the cursor used in collection functions is valid for the collection that each function is applied to.

CLE3005 An identical collection was specified.

Explanation: The addAllFrom() function was called with the same source collection as the target collection.

User Response: Check your program to ensure that the collections are different.

CLE3006 An invalid cursor was specified.

Explanation: The cursor used in collection routines points to an invalid position. There is no collection element at the position specified by the cursor.

User Response: Check your program to ensure that the cursor points to a valid collection element position.

CLE3007 An invalid position was specified.

Explanation: The position specified with a function applied to a collection is invalid for this collection.

User Response: Check your program to ensure that the position is valid for the collection you want to apply the function to.

CLE3008 An invalid replacement was specified.

Explanation: During a collection replaceAt() function, the replacing element had different positioning properties than the element to be replaced.

User Response: Check your program to ensure that the replacing element has the same positioning properties as the element the cursor points to.

CLE3009 A key already exists.

Explanation: A function attempted to add an element to a map or sorted map that already has a different element with the same key.

User Response: Check your program to ensure that the key of the element to be added is different from all existing keys of the elements of the map.

CLE3010 The collection does not contain a key.

Explanation: The collection function elementWithKey() was applied to a collection that does not contain an element with the specified key.

User Response: Check your program to ensure that the collection contains an element with the given key.

CLE3011 This collection is unbounded.

Explanation: The collection function maxNumberOfElements() was applied to a collection that is not bounded.

User Response: Check your program to ensure that the collection is bounded or do not apply the function maxNumberOfElements() to it.

CLE3012 The system is out of memory for collection elements.

Explanation: A function could not obtain the space that it requires.

User Response: Check that the system resources offer enough memory.

CLE3013 A root already exists.

Explanation: A collection function was called for a tree that already has a root.

User Response: Check your program to ensure that the root does not yet exist in your tree.

CLE3014 A cyclic child attachment occurred.

Explanation: A user program tried to attach a subtree to one of its own children.

User Response: Check your program to ensure that you do not try to attach a subtree to one of its own children.

CLE3015 An internal mutex error occurred.

Explanation: You tried to create a Guard and there are no more mutexes available.

User Response: Check the operating system environment parameters. If possible, increase the number of potential concurrent threads or mutexes.

CLE3016 An internal lock error occurred.

Explanation: An error occurred during an internal lock call.

User Response: Check the system environment and reduce the number of threads if possible. Rerun the application.

CLE3017 A time-out occurred.

Explanation: A Guard was requested with a specified time-out value. The internal lock request was not successful.

User Response: Check your application locking sequence, check if all Guard destructors are called, or try to increase the time-out value. Rerun the application.

CLE3018 An internal unlock error occurred.

Explanation: An error occurred during an internal unlock call. The internal lock request was not successful.

User Response: Check the system environment and reduce the number of threads if possible. Rerun the application.

CLE3019 Invalid stream data.

Explanation: The data on the stream is not in the expected format.

User Response: Make sure that the collection stream contains valid data.

CLE3020 Invalid stream helper object.

Explanation: The collection stream helper object is invalid.

User Response: Make sure that the collection stream contains valid data.

CLE3021 The collection is full.

Explanation: The IFullException exception indicates that a function tried to add an element to a bounded collection that was already full. Functions that might cause this exception include add() and addAsFirst().

User Response: Make sure that the application is not trying to add elements to bounded collections that are full.

CLE3022 The collection is not empty.

Explanation: The INotEmptyException exception indicates that a collection operation could not be performed on a collection because the collection is not empty.

User Response: Make sure that the collection is empty, as required.

Chapter 6. z/OS C/C++ USL I/O Stream Class and USL Complex Mathematics Class Library Version 3 Messages

This chapter contains information about the Version 3 USL I/O Stream Class Library and USL Complex Mathematics Class Library messages that are included with z/OS V1R2 IBM Open Class and should not be used as programming interface information.

The following information shows the format of these messages:

Message Format: **CLBnnnn text <&*n*>** where:

nnnn	error message number
text	message which appears on the screen

CLB9000 An attempt to allocate memory has failed.

Explanation: The attempt to obtain memory in order to satisfy the current library request has failed. It cannot be performed on a collection because the collection is not empty.

User Response: Run the program in a larger region or use the HEAP(,,FREE) run-time option instead of the HEAP(,,KEEP) option.

System Action: The requested function will fail.

CLB9001 IOStreams do not support Record Mode I/O.

Explanation: The application is attempting to initialize an IOStreams object to perform Record Mode I/O. IOStream objects do not support Record Mode input and output.

User Response: Remove the ″type=record″ specification from the constructor or open() function call.

System Action: The attempt to initialize the object failed. The program continues to execute.

CLB9002 Too many characters.

Explanation: The application called the form() function with a format specifier string that caused form() to write past the end of the format buffer. form() is an obsolete interface provided in stream.h for compatibility with old code.

User Response: Split the call to the form() function into two or more calls.

System Action: Execution is stopped.

CLB9003 There was a singularity; the application could not take the log of (0.0, 0.0).

Explanation: The application is attempting to take the log of (0.0, 0.0).

User Response: Correct the value passed to the log() function and resubmit.

System Action: Execution is stopped.

CLB9004 The attempt to release the mutex handle failed.

Explanation: There was an internal error: pthread_mutex_destroy() failed.

User Response: Note the return code and error number to identify the cause of the problem and inform IBM C++ Service and Support.

System Action: Execution is stopped.

CLB9005 The attempt to lock the mutex handle failed.

Explanation: There was an internal error: pthread_mutex_lock() failed.

User Response: Note the return code and error number to identify the cause of the problem and inform IBM C++ Service and Support.

System Action: Execution is stopped.

CLB9006 The attempt to unlock the mutex handle failed.

Explanation: Internal error: pthread_mutex_unlock() failed.

User Response: Note the return code and error number to identify the cause of the problem and inform IBM C++ Service and Support.

System Action: Execution is stopped.

Notices

Programming Interface Information

This publication documents *intended* Programming Interfaces that allow the customer to write z/OS C/C++ programs.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States or other countries or both:

AFP	AIX	AT
BookManager	BookMaster	C/370
C/MVS	CICS	CICS/ESA
CT	DB2	DB2 Universal Database

Standards

Extracts are reprinted from IEEE Std 1003.1—1990, IEEE Standard Information Technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C language], copyright 1990 by the Institute of Electrical and Electronic Engineers, Inc.

Extracts are reprinted from IEEE P1003.1a Draft 6 July 1991, Draft Revision to Information Technology—Portable Operating System Interface (POSIX), Part 1: System Application Program Interface (API) [C Language], copyright 1992 by the Institute of Electrical and Electronic Engineers, Inc.

Extracts are reprinted from IEEE Std 1003.2—1992, IEEE Standard Information Technology—Portable Operating System Interface (POSIX)—Part 2: Shells and Utilities, copyright 1990 by the Institute of Electrical and Electronic Engineers, Inc.

Extracts are reprinted from IEEE Std P1003.4a/D6—1992, IEEE Draft Standard Information Technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API)—Amendment 2: Threads Extension [C language], copyright 1990 by the Institute of Electrical and Electronic Engineers, Inc.

For more information on IEEE, visit their web site at `http://www.ieee.org/`.

Extracts from *ISO/IEC 9899:1990* have been reproduced with the permission of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). The complete standard can be obtained from any ISO or IEC member or from the ISO or IEC Central Offices, Case postale 56, CH - 1211 Geneva 20, Switzerland. Copyright remains ISO and IEC. For more information on ISO, visit their web site at `http://www.iso.ch/`.

Bibliography

This bibliography lists the publications for IBM products that are related to the z/OS C/C++ product. It includes publications covering the application programming task. The bibliography is not a comprehensive list of the publications for these products, however, it should be adequate for most z/OS C/C++ users. Refer to *z/OS Information Roadmap*, SA22-7500, for a complete list of publications belonging to the z/OS product.

Related publications not listed in this section can be found on the *IBM Online Library Omnibus Edition MVS Collection*, SK2T-0710, the *z/OS Collection*, SK3T-4269, or on a tape available with z/OS.

z/OS

- *z/OS Introduction and Release Guide*, GA22-7502
- *z/OS Planning for Installation*, GA22-7504
- *z/OS Summary of Message Changes*, SA22-7505
- *z/OS Information Roadmap*, SA22-7500

z/OS C/C++

- *z/OS C/C++ Programming Guide*, SC09-4765
- *z/OS C/C++ User's Guide*, SC09-4767
- *C/C++ Language Reference*, SC09-4815
- *z/OS C/C++ Messages*, GC09-4819
- *z/OS C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS C Curses*, SA22-7820
- *z/OS C/C++ Compiler and Run-Time Migration Guide*, GC09-4913
- *IBM Open Class Library User's Guide*, SC09-4811
- *IBM Open Class Library Reference*, SC09-4812
- *Debug Tool User's Guide and Reference*, SC09-2137
- *Standard C++ Library Reference*, which is available at: `http://www.ibm.com/software/ad/c390/czos/czosdocs.html`

z/OS Language Environment

- *z/OS Language Environment Concepts Guide*, SA22-7567
- *z/OS Language Environment Customization*, SA22-7564
- *z/OS Language Environment Debugging Guide*, GA22-7560
- *z/OS Language Environment Programming Guide*, SA22-7561
- *z/OS Language Environment Programming Reference*, SA22-7562
- *z/OS Language Environment Run-Time Migration Guide*, GA22-7565
- *z/OS Language Environment Writing Interlanguage Applications*, SA22-7563
- *z/OS Language Environment Run-Time Messages*, SA22-7566

Assembler

- *HLASM Language Reference*, SC26-4940
- *HLASM Programmer's Guide*, SC26-4941

COBOL

- *COBOL for OS/390 & VM Compiler and Run-Time Migration Guide*, GC26-4764
- *COBOL for OS/390 & VM Programming Guide*, SC26-9049
- *COBOL for OS/390 & VM Language Reference*, SC26-9046
- *COBOL for OS/390 & VM Diagnosis Guide*, GC26-9047
- *COBOL for OS/390 & VM Licensed Program Specifications*, GC26-9044
- *COBOL for OS/390 & VM Customization under OS/390*, GC26-9045
- *COBOL Millenium Language Extensions Guide*, GC26-9266

PL/I

- *VisualAge PL/I Language Reference*, SC26-9476
- *PL/I for MVS & VM Language Reference*, SC26-3114
- *PL/I for MVS & VM Programming Guide*, SC26-3113
- *PL/I for MVS & VM Compiler and Run-Time Migration Guide*, SC26-3118

VS FORTRAN

- *Language and Library Reference*, SC26-4221
- *Programming Guide*, SC26-4222

CICS

- *CICS Application Programming Guide*, SC34-5702
- *CICS Application Programming Reference*, SC34-5703
- *CICS Distributed Transaction Programming Guide*, SC34-5708
- *CICS Front End Programming Interface User's Guide*, SC34-5710
- *CICS Messages and Codes*, GC33-5716
- *CICS Resource Definition Guide*, SC34-5722
- *CICS System Definition Guide*, SC34-5725
- *CICS System Programming Reference*, SC34-5726
- *CICS User's Handbook*, SX33-6116
- *CICS Family: Client/Server Programming*, SC34-1435
- *CICS Transaction Server for OS/390 Migration Guide*, GC34-5699
- *CICS Transaction Server for OS/390 Release Guide*, GC34-5701
- *CICS Transaction Server for OS/390: Planning for Installation*, GC34-5700

DB2

- *DB2 Administration Guide*, SC26-9931
- *DB2 Application Programming and SQL Guide*, SC26-9933
- *DB2 ODBC Guide and Reference*, GC26-9941
- *DB2 Command Reference*, SC26-9934
- *DB2 Data Sharing: Planning and Administration*, SC26-9935
- *DB2 Installation Guide*, GC26-9936
- *DB2 Messages and Codes*, GC26-9940
- *DB2 Reference for Remote DRDA Requesters and Servers*, SC26-9942
- *DB2 SQL Reference*, SC26-9944

- *DB2 Utility Guide and Reference*, SC26-9945

IMS/ESA

- *IMS/ESA Application Programming: Design Guide*, SC26-8728
- *IMS/ESA Application Programming: Transaction Manager*, SC26-8729
- *IMS/ESA Application Programming: Database Manager*, SC26-8727
- *IMS/ESA Application Programming: EXEC DLI Commands for CICS and IMS*, SC26-8726

QMF

- *Introducing QMF*, GC26-9576
- *Using QMF*, SC26-9578
- *Developing QMF Applications*, SC26-9579
- *Reference*, SC26-9577
- *Installing and Managing QMF on MVS*, SC26-9575
- *Messages and Codes*, SC26-9580

DFSMS

- *z/OS DFSMS Introduction*, SC26-7397
- *z/OS DFSMS: Managing Catalogs*, SC26-7409
- *z/OS DFSMS: Using Data Sets*, SC26-7410
- *z/OS DFSMS Macro Instructions for Data Sets*, SC26-7408
- *z/OS DFSMS Access Method Services*, SC26-7394
- *z/OS DFSMS Program Management*, SC27-1130

INDEX

Program Number: 5694-A01

Printed in the United States of America

GC09-4819-00